Markt&Technik

Mit großem Farbteil

Heiko Knappe

Fraktale Grafik auf dem Amiga

Systeme zwischen Ordnung und Chaos

Auf 3½"-Diskette enthalten:
alle Programme im Quelltext:
Aztec C und Lattice C

Heiko Knappe

Fraktale Grafik auf dem Amiga

Systeme zwischen Ordnung und Chaos

Markt&Technik Verlag AG

CIP-Titelaufnahme der Deutschen Bibliothek

Knappe, Heiko:
Fraktale Grafik auf dem Amiga : Systeme zwischen Ordnung u. Chaos / Heiko Knappe. –
Haar bei München : Markt-u.-Technik-Verl., 1988
ISBN 3-89090-600-1

Die Informationen in diesem Produkt werden ohne Rücksicht auf einen eventuellen Patentschutz veröffentlicht. Warennamen werden ohne Gewährleistung der freien Verwendbarkeit benutzt.
Bei der Zusammenstellung von Texten und Abbildungen wurde mit größter Sorgfalt vorgegangen. Trotzdem können Fehler nicht vollständig ausgeschlossen werden.
Verlag, Herausgeber und Autoren können für fehlerhafte Angaben und deren Folgen weder eine juristische Verantwortung noch irgendeine Haftung übernehmen.

Für Verbesserungsvorschläge und Hinweise auf Fehler sind Verlag und Herausgeber dankbar.

Alle Rechte vorbehalten, auch die der fotomechanischen Wiedergabe und der Speicherung in elektronischen Medien.
Die gewerbliche Nutzung der in diesem Produkt gezeigten Modelle und Arbeiten ist nicht zulässig.

Amiga ist eine Produktbezeichnung der Commodore-Amiga Inc., USA

Amiga-BASIC ist ein eingetragenes Warenzeichen der Microsoft Inc., USA

Lattice C ist ein eingetragenes Warenzeichen der Lattice Corp., USA

Aztec C ist ein eingetragenes Warenzeichen der Manx Software Inc., USA

15 14 13 12 11 10 9 8 7 6 5 4 3 2
91 90 89

ISBN 3-89090-600-1

© 1988 by Markt&Technik Verlag Aktiengesellschaft,
Hans-Pinsel-Straße 2, D-8013 Haar bei München/West-Germany
Alle Rechte vorbehalten
Einbandgestaltung: Grafikdesign Heinz Rauner
Druck: Jantsch, Günzburg
Printed in Germany

Inhalt

Vorwort 7

1 Einleitung 9

2 Erste Experimente 15
2.1 Wo beginnt das Chaos 15
2.2 Erste Auswertungen 18

3 Mathematische Grundlagen
3.1 Was sind komplexe Zahlen? 37
3.2 Neue Rechenregeln? 38
3.3 Rekursive Folgen 41

4 Der Vorstoß ins Chaos
4.1 Die Folge z^2-c - die Mandelbrot-Menge, Julia-Mengen und ihre Verwandten 45
4.1.1 Die Mandelbrot-Menge 46
4.1.2 Zusammenhang zwischen Apfelmännchen und Feigenbaum-Diagramm 49
4.1.3 Geschwindigkeitsoptimierung der Berechnung des Apfelmännchens 50
4.1.4 Julia-Mengen 52
4.1.5 Zusammenhang zwischen Julia-Mengen und Apfelmännchen 57
4.1.6 Symmetrie der Julia-Mengen 60
4.1.7 Weitere Schnitte 61
4.1.8 Mengen höherer Ordnung (z^3-c, z^4-c und z^5-c) 63
4.1.9 Zusammenfassung der Formeln - Hinweise zur Berechnung von Julia-Mengen, Mandelbrot-Mengen und Schnitten 68
4.1.10 Anmerkung zu Berechnungen bei komplexen Mengen 69
4.2 Von der Ebene zur Landschaft - dreidimensionale Untersuchungen der Mandelbrot-Menge 71
4.3 Das Newton-Verfahren - Nullstellensuche einmal ganz anders 78
4.4 Ausblicke 97
4.4.1 Landschaften aus dem Computer 97
4.4.2 Fraktale Computergrafiken - Turtle-Grafik 107

4.4.3 Tapetenmuster 114
4.4.4 Einige Anmerkungen zu den vorgestellten Fraktalen 126
4.4.5 Das seltsame Verhalten fraktaler Kurven - oder Umfang und Inhalt einer Kochschen Schneeflocke 129

5 Wie die Bilder laufen lernen - oder ein Flug ins Apfelmännchen 133

6 Der Fraktal-Generator - Programmbeschreibung zum Programm *fraktal.c* 147

7 Grundlagen dreidimensionaler Grafik 173
7.1 Dreidimensionale Transformationen 173
7.2 Projektion und Perspektive 177
7.3 Unterdrückung verdeckter Oberflächen und Linien 179
7.4 Schattierung 181
7.5 Anlage einer Datenstruktur 184

8 C auf dem Amiga 201
8.1 Inuition - oder ein Computer zum Anfassen 201
8.1.1 Screens 201
8.1.2 Fenster und Intuitionsmeldungen 207
8.1.3 Menüs 213
8.2 Die Grafikfunktionen 218
8.3 DOS 226
8.4 Kurze Zusammenfassung aller wichtiger Funktionen 230
8.5 Das IFF-Format - alles will richtig aufgehoben sein 232

9 Anhang 249
A Diskette zum Buch 259
B Literaturverzeichnis 261
C Daten zu den Bildern 267
D Sachworterklärungen

Vorwort

Das Thema dieses Buches wird sicher vielen exotisch erscheinen. So ziemlich jeder, der sich mit Computern beschäftigt, kennt die Apfelmännchen und seine Verwandten, doch bei der dahinterstehenden Mathematik hört diese Kenntnis meist auf. Das ist auch nicht weiter erstaunlich, denn die Untersuchung von dynamischen Systemen gehört zu den neuesten Forschungsobjekten der Mathematiker. Die Erforschung des Chaos steckt noch in den Kinderschuhen!

Was sind eigentlich computergrafische Experimente?

Seit B. Mandelbrot seine Theorie der Ordnung im Chaos erstmalig vorgestellt hat und es ihm gelang, diese mit Hilfe der Definition von Fraktalen auch mathematisch zu beschreiben, findet in der Mathematik ein Umdenkprozeß statt. Dieses Umdenken ist aber nicht nur auf mathematischem Gebiet im Gange, die Auswirkungen sind auch für die anderen Naturwissenschaften, eigentlich für die gesamte Weltbetrachtung von Bedeutung. Wir stehen vor einer kleinen Revolution in den Wissenschaften, denn wurde bisher davon ausgegangen, daß bei genügender Kenntnis die Welt deterministisch sei, versteht man heute langsam, daß nicht die Ordnung, sondern das Chaos das in der Natur bestimmende Element ist. Meist wird Mathematik als eine starre Wissenschaft betrachtet, in der es nichts mehr zu entdecken gibt. Daß dies nicht stimmt, zeigt die experimentelle Mathematik, die heute mit Methoden der Naturwissenschaften arbeitet. Es wird eine Frage aufgeworfen, die dann durch Verwendung eines Computers zu beantworten versucht wird – ein Bild sagt mehr als tausend Formeln. Der Computer wird zum »Labor«.

Mit dem Amiga haben wir ein ideales »Labor« für diese Experimente zur Verfügung. Sicher, er ist natürlich kein Supercomputer, denn die Berechnungen, die teilweise mehr als zehn Stunden dauern, beanspruchen ihn gewaltig, aber für seinen Preis ist er unschlagbar. Seine schnelle 68000-CPU, sein ausgereiftes Betriebssystem und seine enormen Grafikfähigkeiten machen es möglich.

Den Zugang zur fraktalen Grafik findet man meist über die Faszination an der »inneren Schönheit« dieser Bilder. Zuerst versucht man, die bekannten Entstehungsprozesse nachzuvollziehen, um dann mit immer neuen Variationen dieser das Chaos zu verstehen und zu erforschen. Dies Buch soll den Weg zur eigenen Arbeit vereinfachen und beinhaltet deshalb eine Vielzahl an Beispielprogrammen, die ohne großen Aufwand modifiziert werden können, um damit eine eigene Problemstellung zu bearbeiten.

Mir fiel es zum Teil schwer, mich in die Fachliteratur einzuarbeiten, denn die darin verwendete Mathematik ist nicht leicht zu verstehen. Ich habe versucht, dies Buch nicht

zu einem Fachbuch für Experten zu machen, sondern möglichst allgemein verständlich zu halten. Im Prinzip steht die Produktion von »schönen« Bildern im Vordergrund.

Zum Aufbau dieses Buches

Zu Beginn steht eine allgemeine Einführung zum Thema Fraktale. Im nachfolgenden Abschnitt wird ein einfaches Problem aufgeworfen, dessen Untersuchung uns zum ersten Mal ins Chaos führt – wir lernen die Feigenbaum-Diagramme kennen. Bevor wir uns komplexen Systemen zuwenden, ist noch ein Kapitel den mathematischen Grundlagen gewidmet. Dann begeben wir uns in die faszinierende Welt der komplexen dynamischen Systeme und untersuchen Julia-Mengen, Apfelmännchen und ihre Verwandten. Aber damit wollen wir es nicht belassen: Auch klassische Fraktale, künstliche Landschaften und ähnliches sind interessante Untersuchungsobjekte. Wie wir sehen werden, ist nicht das Chaos der Ausnahmefall, sondern die Ordnung! Um die Berechnungen der räumlichen Darstellungen besser verstehen zu können, wurde ein Kapitel über Grundlagen dreidimensionaler Grafik eingefügt. Dort kann natürlich nur ein kurzer Einblick gegeben werden und dieser Abschnitt ist als Nachschlageteil gedacht. Das letzte Kapitel ist der C-Programmierung auf dem Amiga gewidmet. In diesem Kapitel sind die wichtigsten von mir verwendeten Betriebssystemroutinen und die Probleme, die bei der Programmierung zum Teil aufgetreten sind, erklärt. Dies erscheint mir vor allem in Hinblick auf das Verständnis meiner Programmierweise wichtig zu sein. Auch beinhaltet dieser Abschnitt einige Programmbeispiele, wie zum Beispiel die Initialisierung für Füllroutinen und IFF-Routinen, die problemlos als Module in eigenen Programmen verwendet werden können.

Randbemerkungen

Einen herzlichen Dank an all diejenigen,

- die mein Gejammer über die Arbeit und meine Launen nach einer durchgearbeiteten Nacht ertragen haben,
- die mir auch dann zugehört haben, wenn sie das Thema eigentlich überhaupt nicht interessierte,
- den privaten Radiosendern, die manchmal doch gute Musik brachten,

und meinem Amiga 1000, der mir trotz ständiger Abstürze und tagelangem Streß treu diente.

Mein spezieller Dank gilt meiner Mutter, ohne deren Unterstützung ich dieses Buch niemals fertiggeschrieben hätte.

Heiko Knappe
München

1 Einleitung

Seit der Mensch zu denken gelernt hat, versucht er die Ordnung der Natur zu erforschen und die Erkenntnisse in Gesetze zu fassen. Dies ist auch nicht weiter verwunderlich, denn die Ordnung ist für ihn und für alle höherentwickelten Wesen auf dieser Welt die Grundlage zum Überleben. Unordnung bedroht die Existenz und wird deshalb, wie alles was man nicht versteht und vollständig erfassen kann, als unangenehm empfunden. Daß dieser Grundsatz gerade für die Naturwissenschaften von großer Bedeutung ist, liegt auf der Hand, denn ihre Aufgabe ist es, aus Experimenten und Beobachtungen der Natur auf ihre Gesetze und Modelle zu schließen. Mit wachsendem Fortschrittsglauben in den letzten Jahrhunderten war die Welt für Naturwissenschaftler und Techniker deterministisch, das heißt, sofern man alle Anfangsbedingungen gegeben hat, bis ins letzte Detail vorhersagbar. Wenn Naturerscheinungen nicht erklärt werden konnten, so betrachtete man dies als einen Mangel an Modellvorstellungen, der früher oder später sicher geklärt werden würde. Daß diese Einstellung nicht richtig ist, haben gerade in den letzen Jahren einige Beobachtungen erwiesen. Unser Ziel darf also nicht die Erforschung der Ordnungsprinzipien dieser Welt sein, sondern die Untersuchung des Chaos, denn dies bestimmt die Welt, die Ordnung ist ein Ausnahmefall. Erstaunlich ist, daß die Natur in ihrer Unordnung doch wieder geordnet ist – man spricht deshalb auch von einem »geordneten« bzw. »deterministischen« Chaos. Aber trotz Ordnung im Chaos ist die Natur auf Dauer nicht vorherbestimmbar. Es ist klar, daß wir uns mit einer solchen Einstellung von dem klassischen Verständnis der Naturwissenschaft entfernen und uns vielleicht sogar in philosophische Bereiche vorwagen. Aber ist ein Umdenken angesichts der heutigen Welt mit ihren Katastrophen und Bedrohungen nicht zwingend notwendig? Man muß vorsichtig mit Prognosen sein, aber ich glaube, wir stehen heute in der Naturwissenschaft, der Mathematik und der Technik vor einer kleinen Revolution. Daß diese gerade heute stattfindet, haben wir unserer neuesten technischen Entwicklung zu verdanken – dem Computer. Erst mit seiner Hilfe wird es möglich, Schritte ins »geordnete« Chaos zu unternehmen und Erkenntnisse zu sammeln.

Im folgenden möchte ich einige Beispiele präsentieren, die zeigen sollen, was Chaos ist und wie nahe Ordnung und Unordnung beieinanderliegen.

Betrachten wir die Bewegung der Planeten unseres Sonnensystems um die Sonne, so werden wir zweifelsohne zu dem Schluß kommen, daß diese geordnet ist. Eine Untersuchung des Wachstums von Bakterien zeigt, daß wir, soweit keine äußeren Einflüsse wirksam werden, problemlos vorhersagen können, wie schnell die Vermehrung vor sich gehen wird – also auch hier finden wir Ordnung.

Die Bewegung eines Doppelpendels und einfache Reibungsprobleme sind von uns berechenbar – auch sie sind »in Ordnung«.

Diese geordnete, »beruhigend«-erfaßbare Welt steht auf der einen Seite, auf der anderen das »bedrohliche« Chaos. Zwischen diesen beiden Welten schien lange ein tiefer Spalt zu klaffen, den wir erst heute zu überbrücken beginnen. Unvorhersehbare Katastrophen wie Massenkarambolagen, Flugzeugabstürze und Vulkanausbrüche zeigen uns, daß das Chaos bedrohlich ist und unser Innerstes berührt. Wie unangenehm es ist, sich zum Beispiel einem »Wissenschaos« vor einer Prüfung zu stellen, weiß sicher jeder. Trotz guter Vorbereitung, also geordnetem Lernen des Stoffs, kann durch die Aufregung vor einer Prüfung diese Ordnung in ein Chaos der absoluten Unwissenheit übergehen.

Gerade dieser Zerfall eines scheinbar geordneten Systems durch geringfügige äußere Einflüsse von der Ordnung ins Zufällige ist heute für Forscher auf diesem Gebiet besonders interessant. Sie versuchen dabei, Gesetzmäßigkeiten im Chaos zu erkennen. Deshalb sind folgende Fragen auf ihrer Suche von Bedeutung:

- Was ist Chaos?
- Wie entsteht Chaos?
- Wie kann das Chaos mathematisch erfaßt und beschrieben werden?
- Wo in der Natur treten solche Erscheinungen auf?
- Gibt es Parallelen zwischen verschiedenen Phänomenen?

Besonders wichtig ist: Das Chaos schließt die Ordnung (den Determinismus) nicht aus. Durch die Erkenntnisse der Chaosforschung werfen wir also nicht unsere bisherigen Erfahrungen und Kenntnisse über Bord, sondern erweitern unser Wissen.

Aber wodurch wird ein Vorgang chaotisch, wie geht in ihm die Ordnung verloren?
Verwenden wir eine Videokamera zur Aufzeichnung von Bildern, so erwarten wir die vom Hersteller garantierte Aufnahmequalität. Im Regelfall wird das Ergebnis also nicht »chaotisch« sein, sondern uns ein genaues Abbild der aufgenommenen Objekte liefern. Richten wir nun aber die Kamera auf einen Monitor, der das gerade aufgenommene Bild wiedergibt, passieren unerwartete Dinge. Kippen wir die Kamera ein wenig und manipulieren ihre Empfindlichkeit, entstehen wabbernde Farbmonster und farbige Spiralen, die sich in den Bildschirm zu schrauben scheinen. Seltsam ist, daß diese Bewegungen auch dann entstehen, wenn die Kamera selbst nicht bewegt wird. Hier haben wir wohl einen äußerst komplizierten Rückkopplungsvorgang des elektrischen Systems vor uns. Wie wir noch sehen werden, ist es gerade diese Rückkopplung, die die Grundlage für das Ausbrechen von Unordnung ist. In unserem Fall haben wir diese erzwungen, in der Natur aber ist eine Rückführung die Regel. Die Anzahl von Schädlingen wird zum Beispiel immer von der Population im Vorjahr abhängen.

Daß es oft nicht schwierig ist, ein System ins Chaos zu führen, zeigt das Beispiel eines tropfenden Wasserhahnes. Solange nur wenige Tropfen austreten, fallen diese in regelmäßigen Abständen, → Ordnung. Drehen wir nun den Hahn langsam auf, so werden diese Abstände immer kürzer. Eigentlich erwartet man doch, daß irgenwann die Zeitspanne zwischen den Tropfen sehr klein wird und ein durchgehender Wasserstrahl entsteht.

Dies ist aber nicht der Fall: Kurz bevor der durchgehende Strahl entsteht, fallen Tropfen völlig unterschiedlicher Größe in unregelmäßigen Abständen, → Chaos. Die Unordnung in diesem Fall ist nicht nur einfach zu erzeugen, sondern tritt außerdem völlig unvorhersehbar auf. Geringe Einflüsse haben also weitgreifende Auswirkungen.

Wie oben schon angesprochen, beinhalten Reibungserscheinungen eigentlich keine großen Geheimnisse. Eine sich brechende Welle am Strand oder die Verwirbelung eines Gases beim Ausströmen durch eine Düse zeigen jedoch ein ausgeprägtes chaotisches Verhalten. In diesem Fall sind also die Komplexität des Systems und die Anzahl der Teilchen, die am Reibungsprozeß beteiligt sind, für das Chaos bestimmend. Die einzelnen Wechselwirkungen sind zwar bekannt, aber die mannigfaltigen Rückwirkungen untereinander machen es unmöglich, das Verhalten vorherzubestimmen. Diese Strömungen weisen trotz ihrer Unordnung geordnete Bereiche auf, die alle ein ähnliches Aussehen haben, das abhängig von den physikalischen Gegebenheiten ist. Wir haben damit ein weiteres wichtiges Element des Chaos gefunden – die Selbstähnlichkeit.

Auch in der Biologie beobachten wir ähliche Phänomene. Das Wachstum einer Bakterienkultur folgt, solange keine äußeren Einwirkungen stattfinden, einfach dem Exponentialgesetz. Bei der Vermehrung einer höheren Tierart, beispielsweise den Hasen, deren Bestand von der augenblicklichen Anzahl ihrer Feinde, ihrem Wachstum und ihrer momentanen Anzahl abhängt, wird es schon schwieriger. Je nachdem wie hoch die Wachstumsrate ist, wird das System einem festen Endwert entgegenstreben bzw. ins Chaos übergehen (siehe hierzu »Wo beginnt das Chaos« – Feigenbaum-Diagramme). Ein einfaches Computermodell für diesen Vorgang ist das bekannte »Game of Life«, das durch eine einfache Simulation versucht, die Realität nachzubilden.

Ein ähnliches Chaosverhalten kann man bei sozialen Phänomenen, zum Beispiel dem Ausbruch eines Krieges, beobachten. Auch Menschenansammlungen weisen die typischen Merkmale auf.

Berechnungen, die für Doppelsternsysteme vorgenommen worden sind, zeigen, daß die Planeten unseres Systems gar nicht ordentlich auf bestimmten Bahnen um die Sonne kreisen müssen. Warum befindet sich gerade unser Sonnensystem in solch einer Ordnung? Warum befindet sich die Erde in der richtigen Umlaufbahn um die Sonne und warum ist diese stabil? Welch geringer Anstoß könnte diese Ordnung beenden?

Selbst einfache mechanische Systeme können in chaotische Bewegungen übergehen. Zum Beispiel ein Doppelpendel. Es bewegt sich nach kleinen Anregungen in den erwarteten Bahnen, bei großer Anregung dagegen kann das System ins Chaos übergehen. (Anmerkung: Ein Doppelpendel ist zum Beispiel eine drehbare Scheibe, an deren Rand eine weitere ebenfalls drehbare gelagert ist.)

Abschließend wohl mit das beste Beispiel für ein chaotisches System – das Wetter. Jeder hat sich sicher schon einmal über die Wettervorhersage für den nächsten Tag geärgert. War Sonnenschein angesagt, regnet und stürmt es, oder statt angekündigten Regenfällen (und vielleicht abgesagtem Picknick) scheint die Sonne am wolkenlosen Himmel. In den letzten Jahren können die Meteorologen jedoch dank Computern, vielen Meßstationen und internationaler Zusammenarbeit

das Wetter recht genau vorhersagen. Daß es aber selbst damit niemals gelingen wird, eine hundertprozentige Vorhersage zu treffen, werden wir nach unseren neuen Erkentnissen über chaotische Systeme wohl schon erwartet haben. Die mannigfaltigen Einwirkungen, die das Klima beeinflußen, die vielen Faktoren, die mitbestimmend sind, machen es möglich, daß sich die schönste Vorhersage als völlig falsch erweisen kann. Man spricht hier von dem sogenannten »Schmetterlingseffekt«, denn bereits der Flügelschlag dieses Insekts kann rein theoretisch weitreichende Veränderungen bewirken. Jetzt werden sicher einige denken: »Ja, wenn ich alle Bedingungen und Einflüsse kennen würde, könnte ich auch das Verhalten des Wetters richtig vorhersagen.« Aber die Rückwirkung zufälliger Ereignisse macht selbst bei Kenntnis aller Ausgangsbedingungen die Prognose nach kurzer Zeit unzutreffend. Im Prinzip können wir erst nach der Beobachtung eines solchen Vorganges diesen nachvollziehen und begründen, nicht umgekehrt.

Diese Sammlung von Beispielen ließe sich endlos fortsetzen. Zusammenfassend noch einmal die wichtigsten Kriterien und Eigenschaften für ein kritisches System, das sich chaotisch verhält:

- Rückkopplung der Ausgangswerte
- Komplexität des Systems
- Selbstähnlichkeit
- Grenze zwischen Ordnung und Chaos
- Chaos tritt überall und meistens unerwartet auf.
- kleine Ursache – große Wirkung

Auslösend für den Beginn der Chaosforschung war erstaunlicherweise die Mathematik. Von dieser Geisteswissenschaft erwartet man heute eigentlich keine Innovationen mehr, sondern denkt als Außenseiter, daß sie eine starre, an ihrer Entwicklungsgrenze stehende Wissenschaft sei. Mit der Entdeckung der experimentellen Mathematik betreten die Mathematiker heute völlig unbekannte Gebiete und weisen mit ihrer Erforschung des Chaos auch anderen Wissenbereichen einen neuen Weg in die Zukunft. Die Chaosforschung steht noch ganz am Anfang!

Was ist experimentelle Mathematik? Ihr Hauptmerkmal ist, daß sie im Vergleich zu den herkömmlichen Vorgehensweisen in der Mathematik mit naturwissenschaftlichen Methoden arbeitet. Es wird versucht, Vermutungen oder Annahmen experimentell zu belegen, bzw. umgekehrt die Ergebnisse eines Experiments zu erfassen und dann mathematisch zu beschreiben. Das Untersuchungsobjekt sind zumeist rekursive Folgen, die im Computerlabor berechnet und in Form von Grafiken sichtbar gemacht werden.

An die Öffentlichkeit und zu unerwarteter Popularität gelangte die Chaosforschung wohl einzig und allein wegen dieser schönen und faszinierenden Bilder, die die Resultate der verschiedensten Experimente, Formeln und Auswertungsmethoden wiedergeben. Da diese Bilder von jedermann leicht reproduziert werden können, der einen Computer zu Hause hat, setzte eine regelrechte Euphorie bei der Jagd nach immer neuen Bildformen ein.

Der erste Sturm hat sich heute zwar gelegt, doch die Faszination dieser Bilder ist sicher zeitlos. Außerdem wird die Möglichkeit, selbst an der Erforschung teilzunehmen, durch die ständig wachsende Leistungsfähigkeit der Computer und den Preisverfall auf dem PC-Markt immer weiter wachsen.

Anmerkungen zur Entdeckung

Das Verhalten der rekursiven Folge $z_{n+1} = z_n^2 - c$ in der komplexen Zahlenebene hat der amerikanische Mathematiker B. Mandelbrot erstmals 1980 mit einem Computer untersucht. Seitdem hat die nach ihm benannte Mandelbrot-Menge weltweit großes Aufsehen erregt. Trotz ihrer Einfachheit erzeugt diese Formel reichhaltig strukturierte Grafiken, deren faszinierende Formenvielfalt jeden Betrachter in ihren Bann zieht.

In der Bundesrepublik Deutschland fanden diese Bilder vor allem aufgrund der Ausstellungen des Forschungsteams der Universität Bremen (um Peitgen und Richter) weite Verbreitung. Dieses Forschungsteam hat der Mandelbrot-Menge auch einen Spitznamen verpaßt: das Apfelmännchen. Die einschlägigen Computerzeitschriften entdeckten diese Grafiken erstmals um 1984. Damals war es schwierig, geeignete Literatur zu finden, was nicht weiter verwunderlich ist, wenn man sich überlegt, daß seit der Entdeckung der Mandelbrot-Menge erst fünf Jahre vergangen waren und nur einige Spezialisten sich auf diesem Gebiet auskannten. Die ersten, meist englischsprachigen Fachbücher waren für den mathematischen Laien weitgehend unverständlich. Heute gibt es bereits einige deutschsprachige Bücher, die sich in verständlicher Form mit den verschiedenen Fraktalen befassen.

Mit dem Amiga kam 1986 ein Rechner auf den Markt, der aufgrund seiner grafischen Fähigkeiten geradezu ideal ist, um selber Forschungen auf diesem Gebiet durchzuführen. Dieses Buch soll dem Amiga-Besitzer das Verständnis der komplexen Theorie erleichtern und, ohne große mathematische Vorkenntnisse vorauszusetzen, alle wichtigen Kentnisse vermitteln, die er für seine eigenen Forschungsprojekte braucht. Seit 1980 erlebt die bis dahin als in sich abgeschlossen bekannte Wissenschaft der Mathematik einen Wandel.

Das Vorgehen der Mathematiker nimmt auf dem Gebiet der experimentellen Mathematik immer mehr Methoden der Naturwissenschaftler an. Es werden neue Gedanken und Ideen mit dem Computer überprüft, wobei der umgekehrte Weg gegangen wird: Die berechneten Bilder bedingen neue Feststellungen und Vermutungen, die dann mathematisch bewiesen werden sollen. Diese Arbeitsmethode entspricht dem Vorgehen eines Naturwissenschaftlers und stellt für den Mathematiker etwas völlig Neues dar. Die Problemstellungen, die untersucht werden, sind meist schon bekannt, nur die Auswertung der Ergebnisse mittels Computern und deren grafischen Darstellungsmöglichkeiten eröffnen eine neue Sichtweise. Zum Beispiel ist es schon lange bekannt, daß bei der Untersuchung der Nullstellen einer komplexen Funktion mit dem Newton-Verfahren Probleme in bezug auf die Konvergenz der Folge auftreten können. Die Untersuchung wurde damals nicht weitergeführt, denn der auftretende Rechenaufwand war und ist für einen Menschen zu groß. Erst mit den Computern wurde es möglich, diese Untersuchungen durchzuführen und neue Erkenntnisse zu erhalten.

Das Verdienst B. Mandelbrots ist es, auf diesem Gebiet den Begriff des »Fraktalen« eingeführt zu haben und damit einer neuen Entwicklung die Tore zu öffnen. Ihm gelang es, in den verschiedensten Erscheinungen in Natur und Wissenschaft Parallelen zu finden und diese erstmals zusammenzufassen. Wie vielseitig Fraktale sind, zeigt ihr Auftreten – sie sind einfach überall.

2 Erste Experimente

2.1 Wo beginnt das Chaos?

Bevor wir uns mit dem Auftreten und dem Erscheinungsbild des Chaos beschäftigen wollen, ist es zuerst wichtig zu verstehen, was das eigentlich ist – Chaos. Unter diesem Begriff verstehen wir im allgemeinen Sprachgebrauch Durcheinander, totale Verwirrung und Auflösung jeder Ordnung. Nun gut, welche Vorgänge und Erscheinungen sind eigentlich chaotisch? Hier gibt es viele Beispiele:

- ein unordentliches Zimmer
- Wachstum von Pflanzen
- Verhalten von Strömungen
- Wetterentwicklung

Diese Liste kann man nach einigem Überlegen sicher endlos fortsetzen. Alle diese Erscheinungen haben jedoch eines gemeinsam: Geht man daran, den scheinbar völlig ungeordneten, also chaotischen Zustand zu analysieren, entdeckt man doch eine gewisse Ordnung. Untersuchen wir zum Beispiel Strömungen, so stellen wir schnell fest, daß die entstehenden Verwirbelungen der Flüssigkeit oder des Gases immer wieder ähnlich aussehen und trotz ihrer scheinbar chaotischen Form doch irgendwie geordnet erscheinen.

Die Erforschung des Chaos ist erst in den letzten Jahren ernsthaft angegangen worden, denn erstens glaubten die Wissenschaftler an die Möglichkeit, auch die kompliziertesten Vorgänge bei genügender Kenntnis der Startbedingungen verstehen und vorhersagen zu können, und zweitens mangelte es einfach an Forschungsobjekten. Erst mit dem Aufkommen der Computer ergab sich die Möglichkeit, schon bekannte Probleme in einer vernünftigen Zeit zu untersuchen. Wie heute bekannt ist, haben alle Systeme, die chaotisches Verhalten zeigen, eine eigene innere Ordnung, sozusagen eine Ordnung im Chaos. Gerade diese Eigenschaft macht sie so ansprechend und faszinierend.

Aber wie entsteht Chaos? An welcher Stelle geht die Ordnung in Unordnung über? Gibt es einfache Experimente, mit denen man beginnen kann, um das Chaos zu untersuchen? Die ersten beiden Fragen sind nicht leicht zu beantworten, aber einfache Systeme, die chaotisches Verhalten zeigen, gibt es!

Stellen wir uns doch einmal folgendes Problem: Wir wollen die Vermehrung einer Tierart, zum Beispiel der Hasen untersuchen. Dabei müssen wir folgendes berücksichtigen:

- Die Hasen vermehren sich mit einer Wachstumsrate k und besitzen zu Anfang eine Startpopulation p_0.
- Nun werden die Hasen sich aber nicht einfach beliebig vermehren, sondern ihr Wachstum wird bei einem realistischen Modell von ihrer augenblicklichen Anzahl abhängen. Man kann sich diese Abhängigkeit am besten als Dezimierung der Hasen durch Krankheiten und Bejagung vorstel-

len. Also ist die momentane Population p direkt proportional zu (1-p).

- Andererseits ist das Wachstum sicher von der augenblicklichen Anzahl der Hasen abhängig, also ist p_{n+1} direkt proportional zu p_n.
- Die Anzahl der Hasen wird sich immer mit der gleichen Formel aus der Anzahl im vorherigen Schritt berechnen lassen. Das bedeutet, daß es sich um eine rekursive Folge handelt.

Fassen wir diese Erkenntnisse zusammen, so erhalten wir folgende rekursive Formel:

$$p_{n+1} = p_n + (k * p_n * (1-p_n))$$

Die Population p ist dabei in Prozent anzunehmen. Sinnvolle Werte liegen hier also um den Wert 1.

Nun wird es Zeit, erste Auswertungen vorzunehmen. Das Programm zur Untersuchung der Hasenvermehrung soll uns einen Eindruck über die Folgenglieder vermitteln. Dazu geben wir eine Startpopulation p_0 kleiner als Eins und eine Wachstumsrate k ebenfalls kleiner als Eins vor. Nun werden uns die Werte der jeweiligen Hasenanzahl p_n ausgegeben.

Programm zur Untersuchung von Hasenvermehrung:

```
/* Programm zur Untersuchung von Bevölkerungswachstum (z.B. Hasen) */
ØØ1:
ØØ2:  main(argc,argv)
ØØ3:  int argc;
ØØ4:  char *argv[];
ØØ5:  {
ØØ6:   int i,n;
ØØ7:   float pØ,p,k;
ØØ8:
ØØ9:   if(argc!=2)
Ø1Ø:   {
Ø11:      printf("Eingabeformat: %s pØ,k,n\n",argv[Ø]);
Ø12:      printf("pØ Startpopulation\n");
Ø13:      printf("k  Wachstumsrate\n");
Ø14:      printf("n  Anzahl der Durchlaeufe\n");
Ø15:      exit(1);
Ø16:   }
Ø17:   if((sscanf(argv[1],"%f,%f,%d",&pØ,&k,&n))!=3)
Ø18:   {
Ø19:      printf("Falsche Eingabe! Bitte Wiederholen!\n");
Ø2Ø:      exit(2);
Ø21:   }
Ø22:   printf("pØ=%f,k=%f,n=%d\n\n",pØ,k,n);
Ø23:   for(i=1,p=pØ;i<=n;++i)
Ø24:   {
Ø25:      printf("Schritt: %d -> Population: %f\n",i,p);
Ø26:      p=p*(1+k*(1-p));
Ø27:   }
Ø28:   printf("Ende\n");
Ø29:  }
```

Das Programm liefert bei verschiedener Parameterwahl folgende Ausgaben:

```
pØ=Ø.3ØØØØØ,k=Ø.1ØØØØØ,n=1Ø

Schritt:  1 → Population: Ø.3ØØØØØ
Schritt:  2 → Population: Ø.321ØØØ
Schritt:  3 → Population: Ø.342796
Schritt:  4 → Population: Ø.365325
Schritt:  5 → Population: Ø.388511
Schritt:  6 → Population: Ø.412268
Schritt:  7 → Population: Ø.436498
Schritt:  8 → Population: Ø.461Ø95
Schritt:  9 → Population: Ø.485944
Schritt: 1Ø → Population: Ø.51Ø924

Ende
```

Man erkennt, daß die Anzahl der Hasen monoton steigt. Ein direkter Grenzwert ist nicht zu erkennen. Wie wir aber im nächsten Beispiel sehen werden, ist dies der Wert 1.

```
p0=0.300000,k=0.800000,n=10

Schritt:  1 → Population: 0.300000
Schritt:  2 → Population: 0.468000
Schritt:  3 → Population: 0.667181
Schritt:  4 → Population: 0.844821
Schritt:  5 → Population: 0.949700
Schritt:  6 → Population: 0.987916
Schritt:  7 → Population: 0.997466
Schritt:  8 → Population: 0.999488
Schritt:  9 → Population: 0.999897
Schritt: 10 → Population: 0.999980
Ende
```

Dieses Ergebnis ist eigentlich nicht erstaunlich, denn es ist zu erwarten, daß sich nach einiger Zeit ein natürliches Gleichgewicht einstellt.

Seltsam werden die Ausgabewerte erst bei Wachstumsraten größer als Eins. Plötzlich existiert der Grenzwert (Gleichgewicht!) nicht mehr.

Die Anzahl der Hasen zeigt ein dauerndes, unvorhersehbares Schwanken, und wie wir noch sehen werden: Chaos bricht aus.

```
p0=0.300000,k=1.800000,n=10

Schritt:  1 → Population: 0.300000
Schritt:  2 → Population: 0.678000
Schritt:  3 → Population: 1.070969
Schritt:  4 → Population: 0.934159
Schritt:  5 → Population: 1.044870
Schritt:  6 → Population: 0.960480
Schritt:  7 → Population: 1.028805
Schritt:  8 → Population: 0.975463
Schritt:  9 → Population: 1.018546
Schritt: 10 → Population: 0.984544
Ende
```

Ein weiteres Beispiel belegt unsere Feststellung:

```
p0=0.300000,k=2.800000,n=5

Schritt:  1 → Population: 0.300000
Schritt:  2 → Population: 0.888000
Schritt:  3 → Population: 1.166477
Schritt:  4 → Population: 0.622741
Schritt:  5 → Population: 1.280558
Ende
```

In diesem Bereich oszilliert die Folge also unvorhersehbar, bleibt aber mit ihren Werten zwischen Zwei und Null. Dies ändert sich schlagartig, wählt man eine Wachstumsrate größer als Drei.

Das Chaos geht wieder in Ordnung über, aber die Folge strebt gegen Unendlich und ergibt negative Werte. Diese sind natürlich sinnlos und zeigen, daß das von uns entwickelte Modell in diesem Bereich nicht mehr der Realität entspricht, weniger als keinen Hasen kann es schließlich nicht geben.

```
p0=0.300000,k=5.000000,n=10

Schritt:  1 → Population: 0.300000
Schritt:  2 → Population: 1.350000
Schritt:  3 → Population: -1.012500
Schritt:  4 → Population: -11.200783
Schritt:  5 → Population: -694.492340
Schritt:  6 → Population: -2415765.047073
Schritt:  7 → Population: -29179620742797.850000
Schritt:  8 → Population: -9223371505737304000.000000
Schritt:  9 → Population: -9223371505737304000.000000
Schritt: 10 → Population: -9223371505737304000.000000
Ende
```

Selbst bei dieser einsichtigen Aufgabenstellung und der einfachen Folge erhalten wir unerwartete Ergebnisse.

Wie wir gleich sehen werden, kann die Auswertung auf verschiedene Weise vor sich gehen. Wenn wir die Formel der rekursiven Folge nach unterschiedlichen Kriterien unter-

suchen, wird sich herausstellen, daß das Verhalten der Folge nicht weiter erstaunlich ist. Wir müssen uns nur von unserem anschaulichen Beispiel lösen und die Formel mathematisch betrachten.

2.2 Erste Auswertungen

Die Ausgabe der Ergebnisse als Zahlenwerte ist auf Dauer nicht sehr anschaulich. Auf die Aufmachung kommt es an!

Der erste Schritt in unserem Beispiel ist also die grafische Ausgabe der Folgenwerte. Dabei tragen wir nach rechts die Wachstumsrate k und nach oben die einzelnen Werte der Folge auf. Nun wird das Chaos sichtbar: Es entsteht ein Feigenbaum!

Wenn man zum erstenmal diese Bezeichnung liest und das dazugehörige Bild betrachtet, stellt man sich die Frage, warum in aller Welt diese Grafik »Feigenbaum« heißt. Sie hat doch weder etwas mit Feigen, Bäumen oder gar Feigenbäumen (was immer das sein mag) zu tun. Auch andere Begriffe, denen wir im Laufe dieses Buches noch begegnen werden, sind sehr seltsam und exotisch. Was ist eine Mandelbrot-Ebene, eine Julia-Menge oder ein Apfelmännchen?

Im Falle der Feigenbäume geht die Namensgebung auf einen Physiker namens Mitchell Feigenbaum zurück, der bei der Untersuchung dieser Grafik erstaunliche Entdeckungen gemacht hat. Deshalb heißen die Bilder auch Feigenbaum-Diagramme oder einfach Feigenbäume.

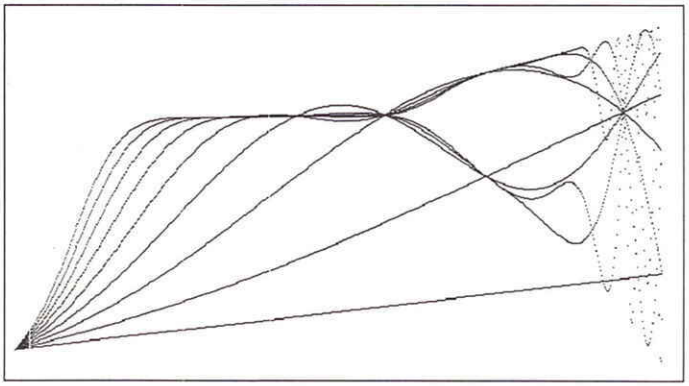
Mit dem folgenden Programm wird ein solcher Feigenbaum berechnet. Um ihn auch nach dem Kompilieren genauer untersuchen zu können, besteht die Möglichkeit, aus dem CLI Werte zu übergeben. Einstellbar ist der Anfang (k_{min}) und das Ende (k_{max}) von dem Bereich der Wachstumsrate, den wir untersuchen wollen. Wie wir bereits bei der vorherigen Auswertung mit Zahlen bemerkt haben, braucht die Folge eine Weile, bis sie sich »eingeschwungen« hat. Deshalb ist es sinnvoll, die ersten Folgewerte zu unterdrücken. Als Parameter können die Anzahl der zu unterdrückenden Werte ($n_\emptyset$) und die Anzahl der danach auszugebenden Werte (n_1) eingegeben werden. Die Startpopulation $p_\emptyset$ ist für alle k-Werte fest gewählt und kann nach dem Kompilieren nicht mehr geändert werden.

Programm: *feigenbaum1.c*

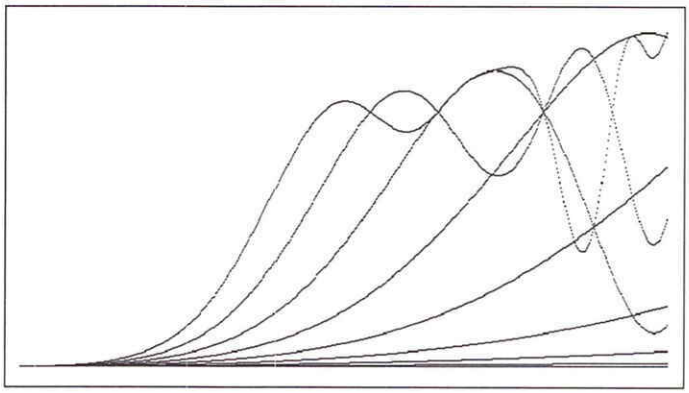
```
 1 /*************************************************************************/
 2 /* feigenbaum1.c                                          Ø1.11.87  */
 3 /* Version 1.1                                                      */
 4 /*                                                                  */
 5 /*                                                                  */
 6 /* Demonstration eines Feigenbaums                                  */
 7 /* mit Eingabemoeglichkeit der Pararmeter aus dem CLI.              */
 8 /*                                                                  */
 9 /* fuer Markt+Technik geschrieben von Heiko Knappe                  */
10 /*                                                                  */
11 /* Komplexe Dynamische Systeme                                      */
12 /* Feigenbaeume - ein Wachstumsproblem                              */
13 /*************************************************************************/
14
15
16 #include <exec/types.h>
17 #include <intuition/intuition.h>
18
19 #define WIDTH  64Ø     /* Screen Modus */
2Ø #define HEIGHT 5ØØ
21 #define DEPTH  2
22 #define FLAGS  HIRES+LACE
23
24 struct IntuitionBase *IntuitionBase;
25 struct GfxBase *GfxBase;
26 struct Screen *MyScreen;
27 struct RastPort *rp;
```

```
struct NewScreen MyNewScreen=
 {0,0,WIDTH,HEIGHT,DEPTH,1,0,FLAGS,CUSTOMSCREEN,0,0,0,0};

int n0=50,n1=100;          /* Offsets fuer die Iteration       */
float p0=.1,p;             /* Start-Population                 */
float kmin=1.9,kmax=3;     /* Relative Wachstums-Rate min/max  */
float st;
float k;                   /* Relative Wachstums-Rate          */
int f,n;                   /* Anzahl der Iterationen           */
BOOL flag=TRUE;            /* dieses Flag wird verwendet, um   */
                           /* noch waehrend der Berechnung das */
                           /* Programm beenden zu koennen.     */


/***********************/
/* Hauptprogramm       */
/***********************/

main(argc,argv)
int argc;
char *argv[];
{
 if((argc!=2)&&(argc!=1))
 {
    printf("Eingabe: %s <n0,n1,kmin,kmax>\n",argv[0]);
    printf(" n0: Anzahl der zu unterdrueckenden Werte\n");
    printf(" n1: Anzahl der auszugebenden Werte\n");
    printf(" [kmin,kmax]: zu untersuchender Bereich\n");
    printf(" p0: Anfangspopulation\n");
    close_things();
 }
 if(argc==2)
    if(sscanf(argv[1],"%d,%d,%f,%f,%f",&n0,&n1,&kmin,&kmax,&p0)!=5)
    {
       printf("falsche Eingabe\n");
       close_things();
    }
 printf("Parameter:\n n0=%d n1=%d\n",n0,n1);
 printf(" kmin=%f kmax=%f p0=%f\n",kmin,kmax,p0);
 st=(kmax-kmin)/WIDTH;
 k=kmin;
 open_things();
 feigenbaum();
 while(MyScreen->TopEdge<100);
 close_things();
}


/***********************/
/* Initialisierung     */
/***********************/

open_things()
{
 if((IntuitionBase=OpenLibrary("intuition.library",0))==0)
    close_things();
 if((GfxBase=OpenLibrary("graphics.library",0))==0)
    close_things();
 if((MyScreen=OpenScreen(&MyNewScreen))==0)
    close_things();
 rp=&(MyScreen->RastPort);
 init_colors();
 Move(rp,10,10);
 Text(rp,"Komplexe Dynamische Systeme",27);
 Move(rp,10,21);
 Text(rp,"p=p*(1+k*(1-p))",15);
}


close_things()
{
 if(IntuitionBase!=0) CloseLibrary(IntuitionBase);
 if(GfxBase!=0) CloseLibrary(GfxBase);
 if(MyScreen!=0) CloseScreen(MyScreen);
 exit();
}


init_colors()
{
 SetRGB4(&(MyScreen->ViewPort),0,0,0,0);
 SetRGB4(&(MyScreen->ViewPort),1,15,0,15);
 SetAPen(rp,1);
}
```

```
112
113
114 /**********************/
115 /* Berechnung         */
116 /**********************/
117
118 feigenbaum()
119 {
120  int x,y;
121
122  for(k=kmin;k<kmax;k+=st)
123  {
124     x=(k-kmin)/st;
125     p=p0;
126     if(MyScreen->TopEdge>100)
127        return(0);
128     for(n=1;n<(n0+n1);++n)
129     {
130        p=p*(1+k*(1-p));
131        if(n>n0)
132        {
133           y=HEIGHT*(1-p/2);
134           if(y<HEIGHT&&y>0) WritePixel(rp,x,y);
135        }
136     }
137  }
138 }
```



2-1 Ausdruck des Programms feigenbaum1.c

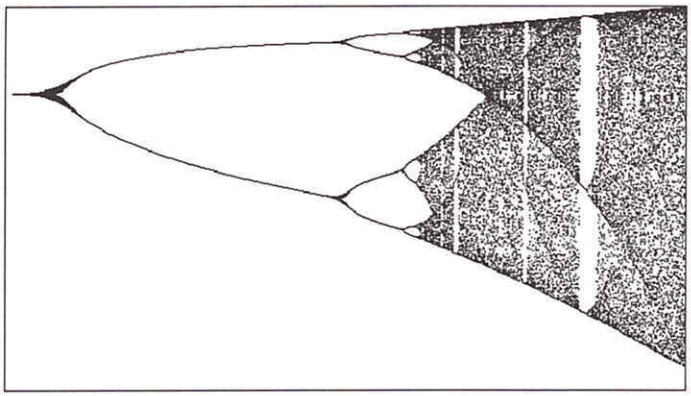


2-2 Ausdruck des Programms feigenbaum1.c

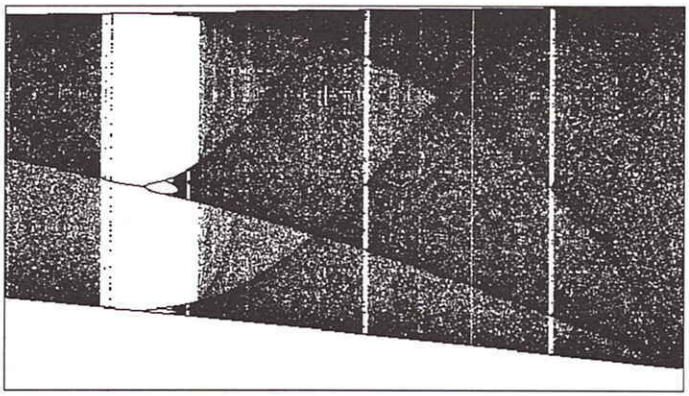
Nun kann's losgehen!

Nach dem Starten des Programms wird ein Screen geöffnet und das Feigenbaum-Diagramm entsteht langsam. Im Bereich kleiner als Eins, um noch einmal zu unserem ersten Beispiel zurückzukehren, wird die Anzahl der Hasen nach einiger Zeit stabil. Das äußert sich dadurch, daß die einzelnen Kurven im Bild gegen eine Gerade streben. Für k-Werte größer als Eins teilt sich die Kurve zuerst einmal, dann wieder und wieder, um schließlich in ein Chaos überzugehen (siehe Abbildungen 2-1 und 2-2). Die Teilung der Kurve zuerst in zwei, dann vier, dann acht Zweige usw. bezeichnet man als periodenverdoppelnde Kaskadenverzweigung. Bei k größer als 2.57 geht das Bild dann in ein völliges Durcheinander von Punkten über, Chaos bricht aus. Bevor wir die hinter diesen Diagrammen stehende Mathematik angehen wollen, zuerst noch einige Bilder, die anschaulich die bisherigen Feststellungen belegen.

Wie in Bild 2-1 zu erkennen, dauert es eine Zeitlang bis die Folgen-Werte »eingeschwungen« sind. Um diesen Effekt zu vermeiden,



2-3 Der Feigenbaum – periodenverdoppelnde Kaskadenverzweigung



2-4 Selbstähnlichkeit – Ordnung im Chaos

wurden in Abbildung 2-3 die ersten 50 Werte unterdrückt und dann erst die folgenden 100 ausgegeben. Wie man erkennt, bleibt das Chaos für k-Werte größer als Zwei immer noch erhalten. Es handelt sich dabei also um keinen Einschwingvorgang. Bei genauer Untersuchung des Bildes erkennt man eine wichtige Eigenschaft aller chaotischen Systeme, die »Ordnung« im Chaos. Betrachtet man nämlich Vergrößerungen einzelner Verzweigungspunkte (siehe Abbildung 2-4), so stellt man fest, daß die entstehenden Bilder zwar alle unterschiedlich, aber dennoch irgendwie ähnlich sind. Diese Eigenschaft nennt man allgemein die Selbstähnlichkeit des Systems.

Unsere bisherige grafische Auswertung hat noch einen Nachteil. Die Anzahl der mehrmals auftretenden Punkte ist nicht erkennbar, denn wenn ein Wert öfter auftrat, haben wir ihn einfach mehrere Male übereinander gezeichnet. Zwei Möglichkeiten dies auszugleichen liegen nahe: Entweder wir versuchen, die Information über die »Tiefe« des Bildes durch Verwendung von Farben auszugleichen, oder wir berücksichtigen sie indem wir ein dreidimensionales »Gebirge« zeichnen. Bei der Auswertung mit Hilfe von Farben tritt folgendes Problem auf: Wir haben nur eine geringe Anzahl von Farben zur Verfügung und können deshalb nicht jedem möglichen Wert genau eine Farbe zuordnen. Da die Zahl, wie oft einzelne Wert auftreten, sehr stark schwankt, ist auch eine grobe Rasterung in 16 Farbbereiche ungenügend. Mit Hilfe der Modulo-Funktion ist dagegen eine bessere Interpretation möglich. Diese Funktion bestimmt den Rest bei einer Division zweier ganzer Zahlen. Es ist also zum Beispiel 12 modulo 5 gleich 2. In unserem Fall bilden wir den Modulo aus »Häufigkeit des Auftretens« und »Anzahl der Farben«. Das erhaltene Ergebnis bestimmt das Farbregister, mit dem der Punkt gezeichnet werden muß. Jetzt ist der oben angeführte Nachteil zwar beseitigt, aber ein neues Problem taucht auf. Verschiedene Häufigkeiten erhalten die gleiche Farbe. Dies macht sich dann im Bild daran bemerkbar, daß periodisch die gleiche Farbe auftritt. Also Vorsicht bei quantitativen Auswertungen. Diese Interpretation ist aber qualitativ sehr aussagekräftig und wird uns noch öfter begegnen. Im Vergleich zum vorherigen Programm können wir nun nicht mehr die Punkte sofort zeichnen, sondern

müssen zuerst alle Werte für eine bestimmte Wachstumsrate k berechnen. Dazu legen wir ein »Raster« für die y-Koordinaten der Punkte an und erhöhen die Rasterelemente entsprechend. Sind alle n1 Schritte durchlaufen, wird die Farbe der einzelnen Punkte, wie oben beschrieben, bestimmt und diese ausgegeben. Die Eingabe der Parameter aus dem CLI ist auch hier möglich und wird genauso wie bei dem Programm *feigenbaum1.c* vorgenommen.

Programm: *feigenbaum2.c*

```
/*************************************************************************/
/* feigenbaum2.c                                              Ø1.11.87*/
/* Version 1.1                                                         */
/*                                                                     */
/*                                                                     */
/* Demonstration eines Feigenbaums                                     */
/* unter  farblicher Auswertung der Hauefigkeiten und Eingabe-         */
/* möglichkeit der Parameter aus dem CLI.                              */
/*                                                                     */
/* fuer Markt+Technik geschrieben von Heiko Knappe                     */
/*                                                                     */
/* Komplexe Dynamische Systeme                                         */
/* Feigenbaeume - ein Wachstumsproblem                                 */
/*************************************************************************/


#include <exec/types.h>
#include <intuition/intuition.h>

#define WIDTH  64Ø          /* Screen Modus */
#define HEIGHT 5ØØ
#define DEPTH  3
#define FLAGS  HIRES+LACE

struct IntuitionBase *IntuitionBase;
struct GfxBase *GfxBase;
struct Screen *MyScreen;
struct RastPort *rp;
struct NewScreen MyNewScreen=
 {Ø,Ø,WIDTH,HEIGHT,DEPTH,1,Ø,FLAGS,CUSTOMSCREEN,Ø,Ø,Ø,Ø};

int anzcol;
int nØ=5Ø,n1=1ØØ;          /* Offsets fuer die Iteration       */
float pØ=.1,p;             /* Start-Population                 */
float kmin=1.9,kmax=3;     /* Relative Wachstums-Rate min/max  */
float st;
float k;                   /* Relative Wachstums-Rate          */
int n;                     /* Anzahl der Iterationen           */


/***********************/
/* Hauptprogramm       */
/***********************/

main(argc,argv)
int argc;
char *argv[];
{
 int i;

 if((argc!=2)&&(argc!=1))
 {
    printf("Eingabe: %s <nØ,n1,kmin,kmax,pØ>\n",argv[Ø]);
    printf(" nØ: Anzahl der zu unterdrueckenden Werte\n");
    printf(" n1: Anzahl der auszugebenden Werte\n");
    printf(" [kmin,kmax]: zu untersuchender Bereich\n");
    printf(" pØ: Startpopulation");
    close_things();
 }
 if(argc==2)
    if(sscanf(argv[1],"%d,%d,%f,%f,%f",&nØ,&n1,&kmin,&kmax,&pØ)!=5)
    {
       printf("falsche Eingabe\n");
       close_things();
    }
 printf("Parameter:\n nØ=%d n1=%d\n",nØ,n1);
 printf(" kmin=%f kmax=%f pØ=%f\n",kmin,kmax,pØ);
 for(i=1,anzcol=1;i<=DEPTH;++i,anzcol*=2)  /* Anzahl der Farben
                                              berechnen          */
 st=(kmax-kmin)/WIDTH;
```

```
70   k=kmin;
71   open_things();
72   feigenbaum();
73   while(MyScreen->TopEdge<100);
74   close_things();
75  }
76
77
78  /***********************/
79  /* Initialisierung     */
80  /***********************/
81
82  open_things()
83  {
84   if((IntuitionBase=OpenLibrary("intuition.library",0))==0)
85      close_things();
86   if((GfxBase=OpenLibrary("graphics.library",0))==0)
87      close_things();
88   if((MyScreen=OpenScreen(&MyNewScreen))==0)
89      close_things();
90   rp=&(MyScreen->RastPort);
91   init_colors();
92   SetAPen(rp,3);
93   Move(rp,10,10);
94   Text(rp,"Komplexe Dynamische Systeme",27);
95   Move(rp,10,21);
96   Text(rp,"p=p*(1+k*(1-p))",15);
97  }
98
99
100 close_things()
101 {
102  if(IntuitionBase!=0) CloseLibrary(IntuitionBase);
103  if(GfxBase!=0) CloseLibrary(GfxBase);
104  if(MyScreen!=0) CloseScreen(MyScreen);
105  exit();
106 }
107
108
109 init_colors()
110 {
111  int i;
112
113  for(i=0;i<anzcol;++i)
114     SetRGB4(&(MyScreen->ViewPort),i,(int)((float)i/(float)anzcol*16),0,
115           (int)((1-(float)i/(float)anzcol)*16));
116 }
117
118
119 /***********************/
120 /* Berechnung          */
121 /***********************/
122
123 feigenbaum()
124 {
125  int x,y,anz[HEIGHT];
126  BYTE color;
127
128  for(k=kmin;k<kmax;k+=st)
129  {
130     if(MyScreen->TopEdge>100)
131        return(0);
132     p=p0;
133     x=(k-kmin)/st;
134     for(y=0;y<HEIGHT;++y)          /* Loeschen          */
135        anz[y]=0;
136     for(p=p0,n=0;n<(n0+n1);++n)    /* Werte berechnen   */
137     {
138        p=p*(1+k*(1-p));
139        y=HEIGHT*(1-p/2);
140        if(n>=n0) ++anz[y];         /* Unterdruecken!    */
141     }
142     for(y=0;y<HEIGHT;++y)          /* Ausgabe           */
143     {
144        if(anz[y]!=0) color=(anz[y]%(anzcol-1))+1;
145        else color=0;
146        SetAPen(rp,color);
147        if((y>0)&&(y<HEIGHT)&&(color!=0)) WritePixel(rp,x,y);
148     }
149  }
150 }
```

Wie oben bereits angesprochen, erhält man auch bei einer dreidimensionalen Auswertung eines Feigenbaumes interessante Bilder. Es ist sinnvoll, auch hier mit Farben zu arbeiten. Dazu noch einige Bemerkungen: Die Höhe des Gebirges wird aus der absoluten Häufigkeit bestimmt, und die Farbe des Punktes errechnet man genauso wie im letzten Beispiel. Um eine räumliche Tiefe zu erzielen, zeichnet man von einer projizierten Ebene eine Gerade senkrecht nach oben, deren Länge proportional zu der Häufigkeit der Werte in diesem Punkt ist. Um eine »Beleuchtung« und damit ein räumliches Gebilde zu erhalten, sollte man am oberen Ende der Geraden einen Punkt mit der zur errechneten komplementären Farbe setzen. Wurde also zum Beispiel Farbregister 9 aus 16 möglichen (gezählt von Ø bis 15) berechnet, so ist mit komplementärer Farbe Farbregister (15-9)=6 gemeint. Die Berechnungen können natürlich, wie im letzten Beispiel, nur zeilenweise ausgegeben werden. Bei dieser Auswertung gleichen wir auch den Nachteil des periodischen Auftretens gleicher Farben aus.

Programm: *feigenbaum 3.c*

```
/*************************************************************************/
/* feigenbaum3.c                                              Ø1.11.87 */
/* Version 1.1                                                         */
/*                                                                     */
/*                                                                     */
/* Demonstration eines Feigenbaums                                     */
/* unter farblicher Auswertung der Hauefigkeiten, 3 dimensionaler      */
/* Darstellung und Eingabemoeglichkeit der Parameter aus dem CLI.      */
/*                                                                     */
/* fuer Markt+Technik geschrieben von Heiko Knappe                     */
/*                                                                     */
/* Komplexe Dynamische Systeme                                         */
/* Feigenbaeume - ein Wachstumsproblem                                 */
/*************************************************************************/

#include <exec/types.h>
#include <intuition/intuition.h>

#define WIDTH  64Ø     /* Screen Modus */
#define HEIGHT 5ØØ
#define DEPTH  3
#define FLAGS  HIRES+LACE

/* Definition der 'Feldgroesse' fuer die 3-d Ausgabe */
#define FWIDTH   4ØØ
#define FHEIGHT  2ØØ
#define FXOFFSET 2Ø
#define FYOFFSET 15Ø
#define OFF      1Ø

struct IntuitionBase *IntuitionBase;
struct GfxBase *GfxBase;
struct Screen *MyScreen;
struct RastPort *rp;
struct NewScreen MyNewScreen=
 {Ø,Ø,WIDTH,HEIGHT,DEPTH,1,Ø,FLAGS,CUSTOMSCREEN,Ø,Ø,Ø,Ø};

int anzcol;
int nØ=5Ø,n1=2ØØ;          /* Offsets fuer die Iteration       */
float pØ=.1,p;             /* Start-Population                 */
float kmin=1.9,kmax=3;     /* Relative Wachstums-Rate min/max  */
float st;
float k;                   /* Relative Wachstums-Rate          */
int n;                     /* Anzahl der Iterationen           */


/***********************/
/* Hauptprogramm       */
/***********************/

main(argc,argv)
int argc;
char *argv[];
{
 int i;

```

```
58  if((argc!=2)&&(argc!=1))
59  {
60     printf("Usage: %s <nØ,n1,kmin,kmax,pØ>\n",argv[Ø]);
61     close_things();
62  }
63  if(argc==2)
64     if(sscanf(argv[1],"%d,%d,%f,%f,%f",&nØ,&n1,&kmin,&kmax,&pØ)!=5)
65     {
66        printf("falsche Eingabe\n");
67        close_things();
68     }
69  printf("Parameter:\n nØ=%d n1=%d\n kmin=%f kmax=%f pØ=%f\n",
7Ø        nØ,n1,kmin,kmax,pØ);
71  for(i=1,anzcol=1;i<=DEPTH;++i,anzcol*=2)  /* Anzahl der Farben
                                                 berechnen          */
72  st=(kmax-kmin)/FWIDTH;
73  k=kmin;
74  open_things();
75  feigenbaum();
76  while(MyScreen->TopEdge<1ØØ);
77  close_things();
78 }
79
8Ø
81 /***********************/
82 /* Initialisierung     */
83 /***********************/
84
85 open_things()
86 {
87  if((IntuitionBase=OpenLibrary("intuition.library",Ø))==Ø)
88     close_things();
89  if((GfxBase=OpenLibrary("graphics.library",Ø))==Ø)
9Ø     close_things();
91  if((MyScreen=OpenScreen(&MyNewScreen))==Ø)
92     close_things();
93  rp=&(MyScreen->RastPort);
94  init_colors();
95  SetAPen(rp,3);
96  Move(rp,1Ø,1Ø);
97  Text(rp,"Komplexe Dynamische Systeme",27);
98  Move(rp,1Ø,21);
99  Text(rp,"p=p*(1+k*(1-p))",15);
1ØØ }
1Ø1
1Ø2
1Ø3 close_things()
1Ø4 {
1Ø5  if(IntuitionBase!=Ø) CloseLibrary(IntuitionBase);
1Ø6  if(GfxBase!=Ø) CloseLibrary(GfxBase);
1Ø7  if(MyScreen!=Ø) CloseScreen(MyScreen);
1Ø8  exit();
1Ø9 }
11Ø
111
112 init_colors()
113 {
114  int i;
115
116  for(i=Ø;i<anzcol;++i)
117     SetRGB4(&(MyScreen->ViewPort),i,(int)((float)i/(float)anzcol*16),Ø,
118             (int)((1-(float)i/(float)anzcol)*16));
119 }
12Ø
121
122 /***********************/
123 /* Berechnung          */
124 /***********************/
125
126 feigenbaum()
127 {
128  int i,x,ys,ye,anz[FHEIGHT];
129  short color;
13Ø
131  for(k=kmin;k<kmax;k+=st)
132  {
133     if(MyScreen->TopEdge>1ØØ)
134        return(Ø);
135     p=pØ;
136     for(i=Ø;i<FHEIGHT;++i)                 /* Loeschen           */
137        anz[i]=Ø;
138     for(p=pØ,n=Ø;n<(nØ+n1);++n)            /* Werte berechnen    */
139     {
```

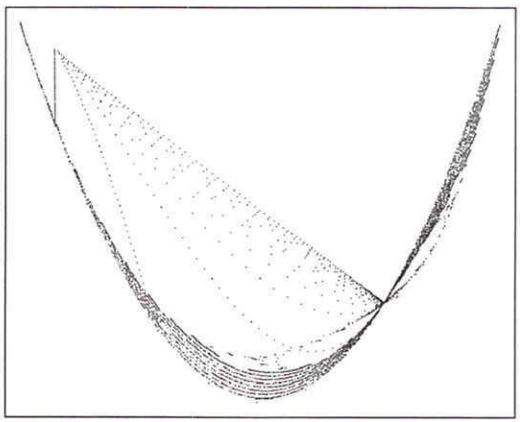
```
14Ø          p=p*(1+k*(1-p));
141          i=FHEIGHT*(1-p/2);
142          if(n>=nØ&&i<FHEIGHT&&i>=Ø) ++anz[i]; /* Unterdruecken     */
143       }
144       for(i=Ø;i<FHEIGHT;++i)                  /* Ausgabe          */
145       {
146          if(anz[i]!=Ø) color=anz[i]%(anzcol-1)+1;
147          else color=Ø;
148          SetAPen(rp,color);
149          x=(k-kmin)/st+FXOFFSET+(FHEIGHT-i);
15Ø          ys=FYOFFSET+i;
151          ye=ys-anz[i]-OFF;
152          if(ys>HEIGHT) ys=HEIGHT;
153          if((x>Ø)&&(x<WIDTH)&&(ys>Ø)&&(ys<HEIGHT)&&(color!=Ø))
154          {
155             Move(rp,x,ys);
156             if(ye<Ø) ye=Ø;
157             Draw(rp,x,ye);
158             SetAPen(rp,anzcol-color);
159             WritePixel(rp,x,ye);
16Ø          }
161       }
162    }
163 }
```

Wie die letzten Beispiele gezeigt haben, ist es nötig, geeignete Auswertungskriterien zu finden, damit die erhaltenen Ergebnisse einen möglichst guten Eindruck vermitteln. Manchmal ist das gar nicht so einfach! Dies soll kein Lehrbuch sein, und hier sollen auch keine Aufgaben gestellt werden. Aber ein Vorschlag: Gehen Sie doch auf Entdeckungsreise! Untersuchen Sie den Chaosbereich und erzeugen Bilder, die wahrscheinlich noch niemand vor Ihnen gesehen hat. Die letzten drei Programme bieten schon einige Möglichkeiten, in die Welt der chaotischen Systeme vorzudringen.

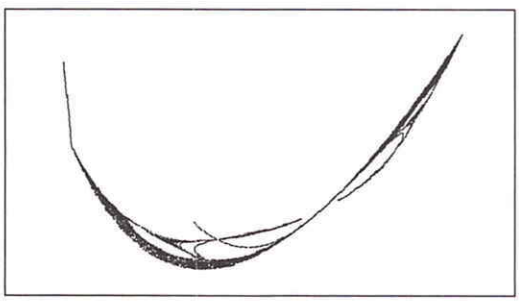
Bisher haben wir das Verhalten der komplexen Folge

$$p_{n+1} = p_n * (1 + k*(1-p_n))$$

in bezug auf die Abhängigkeit der auftretenden Folgen-Werte von der Wachstumsrate k untersucht. Die Folge hängt aber nicht nur von k, sondern auch von dem Startwert $p_Ø$ ab. Zeichnen wir nun doch einmal nach rechts die Werte p_n und nach oben p_{n+1} und wählen gleichzeitig k fest. Wie wir erwarten, entsteht für kleine k-Werte eine Gerade. Dann beginnt die kontinuierlich wachsende Kurve auf einmal zuerst an zwei, dann vier usw. Stellen zu wachsen. Im Chaos-Bereich schließlich tauchen wahllos an verschiedenen Stellen Punkte auf, die sich nach und nach zu einer Parabel-ähnlichen Figur verdichten (siehe Abbildungen 2-5 und 2-6). Um all



2-5 Ausdruck des Programms feigenbaum4.c



2-6 Die innere Struktur der Parabel

diese Kurven gleichzeitig vergleichen zu können, ist es sinnvoll, mehrere von ihnen in ein Bild zu zeichnen. Wieder ist die erstaunliche Ordnung, die Selbstähnlichkeit und die Aufteilung zu erkennen. Vergleicht man die beim vorigen Auswerte-Typ entstandenen Bilder mit den jetzigen, so ist es sicher nicht auf den ersten Blick einzusehen, daß beide durch dieselbe Folge entstanden sind.

Programm: *feigenbaum4.c*

```
/*****************************************************************/
/* feigenbaum4.c                                        Ø1.11.87*/
/* Version 1.1                                                  */
/*                                                              */
/*                                                              */
/* Auswertung der Folge p(n+1)=p(n)*(1+k*(1-p(n))) durch Auf-   */
/* tragen von p(n) nach Rechts und p(n+1) nach Unten mit        */
/* Eingabemöglichkeit der Pararmeter aus dem CLI.               */
/*                                                              */
/* fuer Markt+Technik geschrieben von Heiko Knappe              */
/*                                                              */
/* Komplexe Dynamische Systeme                                  */
/* Feigenbaeume - ein Wachstumsproblem                          */
/*****************************************************************/


#include <exec/types.h>
#include <intuition/intuition.h>

#define WIDTH  64Ø        /* Screen Modus                       */
#define HEIGHT 5ØØ
#define DEPTH  2
#define FLAGS  HIRES+LACE

struct IntuitionBase *IntuitionBase;
struct GfxBase *GfxBase;
struct Screen *MyScreen;
struct RastPort *rp;
struct NewScreen MyNewScreen=
 {Ø,Ø,WIDTH,HEIGHT,DEPTH,1,Ø,FLAGS,CUSTOMSCREEN,Ø,Ø,Ø,Ø};

int nØ=Ø,n1=1ØØ;          /* Offsets fuer die Iteration       */
float pØ=.1,pneu,palt;    /* Start-Population / Population    */
float k,kmin=Ø,kmax=3,kst; /* Bereich der rel. Wachstums-Rate */
int st=1Ø;                /* Anzahl der Unterteilungen        */
int n;                    /* Anzahl der Iterationen           */


/***********************/
/* Hauptprogramm       */
/***********************/

main(argc,argv)
int argc;
char *argv[];
{
 if((argc!=2)&&(argc!=1))
 {
    printf("Eingabe: %s <nØ,n1,pØ,kmin,kmax,st>\n",argv[Ø]);
    printf(" nØ: Anzahl der zu unterdrueckenden Werte\n");
    printf(" n1: Anzahl der auszugebenden Werte\n");
    printf(" pØ: Anfangswert\n");
    printf(" [kmin,kmax]: Wertebereich der Wachstumsrate\n");
    printf(" st: Anzahl der Schritte\n");
    close_things();
 }
 if(argc==2)
    if(sscanf(argv[1],"%d,%d,%f,%f,%f,%d",&nØ,&n1,&pØ,&kmin,&kmax,&st)!=6)
    {
       printf("falsche Eingabe\n");
       close_things();
    }
 printf("Parameter:\n nØ=%d n1=%d\n pØ=%f\n",nØ,n1,pØ);
 printf(" kmin=%f kmax=%f st=%d\n",kmin,kmax,st);
 k=kmin;
 kst=(kmax-kmin)/(float)st;
 open_things();
 for(;;)
 {
  if(k<kmax)
  {
     k+=kst;
```

```
73        rech();
74     }
75     if(MyScreen->TopEdge>1ØØ)
76        close_things();
77     }
78  }
79
8Ø
81  /***********************/
82  /* Initialisierungs-   */
83  /* routinen            */
84  /***********************/
85
86  open_things()
87  {
88     IntuitionBase=OpenLibrary("intuition.library",Ø);
89     if(IntuitionBase==Ø)
9Ø        close_things();
91     GfxBase=OpenLibrary("graphics.library",Ø);
92     if(GfxBase==Ø)
93        close_things();
94     MyScreen=OpenScreen(&MyNewScreen);
95     if(MyScreen==Ø)
96        close_things();
97     rp=&(MyScreen->RastPort);
98     init_colors();
99     Move(rp,1Ø,1Ø);
1ØØ    Text(rp,"Komplexe Dynamische Systeme",27);
1Ø1    Move(rp,1Ø,21);
1Ø2    Text(rp,"p=p*(1+k*(1-p))",15);
1Ø3  }
1Ø4
1Ø5
1Ø6  close_things()
1Ø7  {
1Ø8     if(IntuitionBase!=Ø) CloseLibrary(IntuitionBase);
1Ø9     if(GfxBase!=Ø) CloseLibrary(GfxBase);
11Ø     if(MyScreen!=Ø) CloseScreen(MyScreen);
111     exit();
112  }
113
114
```

```
115  init_colors()
116  {
117     SetRGB4(&(MyScreen->ViewPort),Ø,Ø,Ø,Ø);
118     SetRGB4(&(MyScreen->ViewPort),1,15,Ø,15);
119     SetAPen(rp,1);
12Ø  }
121
122
123  /***********************/
124  /* Berechnung          */
125  /***********************/
126
127  rech()
128  {
129     int x,y;
13Ø
131     palt=pØ;
132     for(n=Ø;n<(nØ+n1);++n)
133     {
134        pneu=palt*(1+k*(1-palt));
135        if(n>=nØ)
136        {
137           x=.7*palt*WIDTH;
138           y=.7*pneu*HEIGHT;
139           if(x>Ø&&x<WIDTH&&y<HEIGHT&&y>Ø) WritePixel(rp,x,y);
14Ø        }
141        palt=pneu;
142     }
143  }
```

Die bisherigen Arten der Auswertung haben uns eigentlich nur gezeigt, daß ein einfaches System ziemlich erstaunliche Verhaltensweisen aufweist. Wir haben erkannt, daß es in Chaos ausbrechen kann, daß dieses strukturiert ist, und haben die Selbstähnlichkeit untersucht. Unser Vorgehen zeigte uns dies jedoch nur qualitativ, und viele Fragen sind offen geblieben:

- Für welche Anfangswerte von `p` und `k` konvergiert die Folge überhaupt?

- Wann bricht das Chaos aus?
- Gibt es ein mathematisches Modell, das dieses Verhalten verständlich werden läßt?

Im folgenden wollen wir versuchen, diese Fragen zu beantworten, die Feigenbaumdiagramme besser zu verstehen und so den ersten Schritt zum Verständnis des Chaos zu machen.

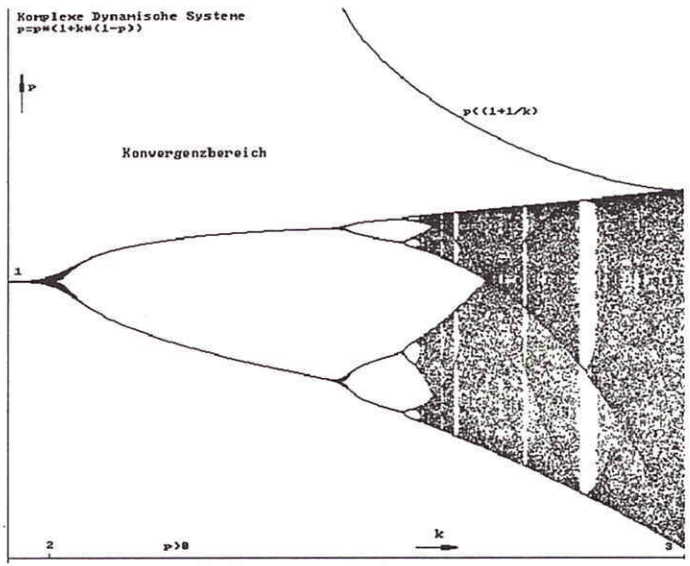
Bereits bei der zahlenmäßigen Auswertung haben wir festgestellt, daß die Folge für negative Werte ganz sicher divergiert. Eine Grenze des Bereichs ist also die Gerade $p=0$. Für ein konvergentes Verhalten muß jedes Folgenglied größer als Null sein. Wir können die Iterationsformel $p_{n+1}=p_n+(k*p_n*(1-p_n))$ folgendermaßen umformen, um ihren Konvergenzbereich zu erfassen:

$$p_n*(1+k*(1-p_n)) > 0 \text{ (für } p_n>0)$$

Nach kurzer Umformung erhalten wir dann die notwendige Konvergenzbedingung:

$$p_n < \left(1 + \frac{1}{k}\right)$$

Dies ist eine nach oben verschobene Hyperbel, die die zweite Grenze des Einzugsgebiets bildet. Starten wir also mit einem Wert p_0 zwischen der Geraden $p=0$ und der Hyperbel $p<\left(1 + \frac{1}{k}\right)$, so können wir erwarten, daß die Folge konvergiert (siehe Abbildung 2-7). Es ist natürlich nicht ganz richtig von konvergentem Verhalten zu sprechen, denn im Bereich für Werte von k kleiner als drei wissen wir ja, daß sich die Folge keinem festen Wert nähert. Es ist besser zu sagen: Für Startwerte aus dem Einzugsbereich »landen« wir sicher bei dem Attraktor, dem Feigenbaum. Das gesamte Feigenbaum-Diagramm kann als »Grenzwert« gesehen werden, gegen den die Folge konvergiert.



2-7 Für welche Startwerte konvergiert die Folge?

Damit ist die erste Frage beantwortet. Versuchen wir nun doch einmal die Struktur des Diagramms zu untersuchen. Wie wir wissen, verzweigt die Kurve erst in zwei, dann vier, dann acht usw. Äste. Dieses Phänomen wird als doppelte Kaskadenverzweigung bezeichnet. Wir erkennen, daß die Verzweigungen in immer kürzeren Abständen auftreten, aber ist es möglich, den k-Wert an der nächsten Verzweigung vorherzusagen? Mitchel Feigenbaum kam als erster auf den Gedanken, das Auftreten dieser Aufteilungen zu quantisieren. Er hat bewiesen, daß für die Aufteilungspunkte k_i folgender Zusammenhang besteht:

$$d = \lim_{n\to\infty}\left(\frac{(k_n-k_{n-1})}{(k_{n+1}-k_n)}\right) = 4.6692016091029997\ldots$$

Die Folge

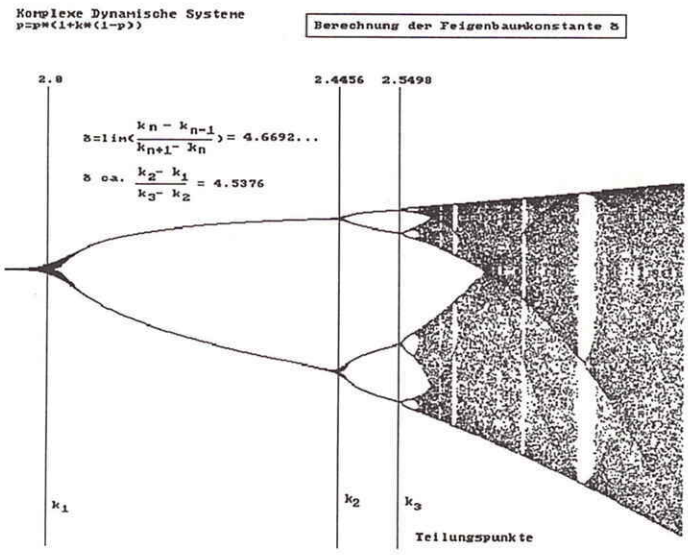
$$\frac{(k_n-k_{n-1})}{(k_{n+1}-k_n)}$$

strebt also gegen einen festen Wert. Wie sich herausgestellt hat, tritt dieser Wert nicht nur hier auf, man kann ihn als eine Art Naturkonstante der Mathematik sehen.

Nun versuchen wir doch einmal zu überprüfen, ob dieser Wert korrekt ist. Dazu lesen wir aus einem von uns gezeichneten Feigenbaum-Diagramm den ersten, zweiten und dritten k-Wert ab, bei dem Verzweigungen auftreten. Dazu empfiehlt es sich natürlich, den interessanten Bereich möglichst groß zu zeichnen, um besser ablesen zu können. Daß unser nun berechneter Wert nur eine grobe Näherung sein kann, ist zu erwarten, denn die Konstante d wird erst für n gegen Unendlich angenommen. Wir erhalten:

$k_1 = 2.\emptyset$
$k_2 = 2.4456$
$k_3 = 2.5498$

und damit d = 4.5376. Dies Ergebnis ist wie erwartet ungenau, und nur die erste Stelle stimmt überein (siehe Abbildung 2-8).



2-8 Berechnung (grobe Näherung!) der Naturkonstante »d«

Die dem Feigenbaum-Diagramm zugrunde liegende Gleichung, die wir aus unserem Hasenmodell abgeleitet haben, wurde bereits 1845 von dem Mathematiker Verhulst formuliert. Deshalb wird diese Gleichung auch als Verhulst-System und der Feigenbaum als Verhulst-Attraktor bezeichnet.

Bisher haben wir die Folge in der k-p-(Feigenbaum-Diagramm) und in der p_n-p_{n+1}-Ebene (k konstant) untersucht. Die Verhulst-Formel hängt aber von drei Parametern ab, nach denen wir sie untersuchen können.

$$p_{n+1} = p_n * (1+k*(1-p_n))$$

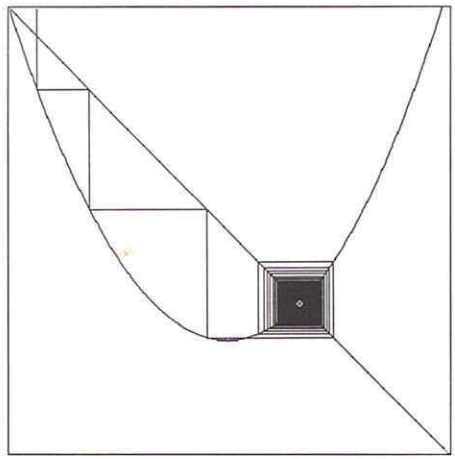
Dabei sind p_n und k die Anfangsbedingungen und p_{n+1} das daraus folgende Ergebnis. Als dritte Variante wollen wir nun versuchen, das Verhalten als grafische Iteration darzustellen. Dazu schreiben wir die Gleichung:

$$p_{n+1} = -k*p_n^2 + (1+k)*p_n.$$

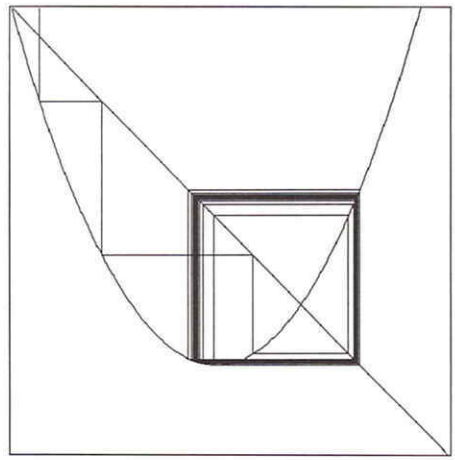
Sie beschreibt also eine nach unten geöffnete Parabel im ersten Quadranten mit Nullstellen bei 0 und $1 + \frac{1}{k}$ und einem Scheitelpunkt bei $\frac{1}{2}*\left(1+\frac{1}{k}\right)$ wenn nach oben p_{n+1} und nach rechts p_n aufgetragen wird. Für unsere Auswertung wählen wir k konstant und einen Startwert $p_\emptyset$. Nun gehen wir von $p_\emptyset$ aus solange nach oben, bis wir die Parabel erreichen. Damit haben wir den Wert p_1 erhalten. Um ihn richtig zu übertragen, gehen wir in horizontaler Richtung, bis die winkelhalbierende $p_{n+1} = p_n$ erreicht ist. Jetzt wiederholt sich der vorletzte Schritt. Wir gehen in vertikaler Richtung, bis wir wieder auf die Parabel stoßen . . .

Wiederholen wir dieses Verfahren mehrere Male hintereinander, erkennen wir an den Resultaten, gegen welche Werte die Folge strebt bzw. welche Werte angenommen werden.

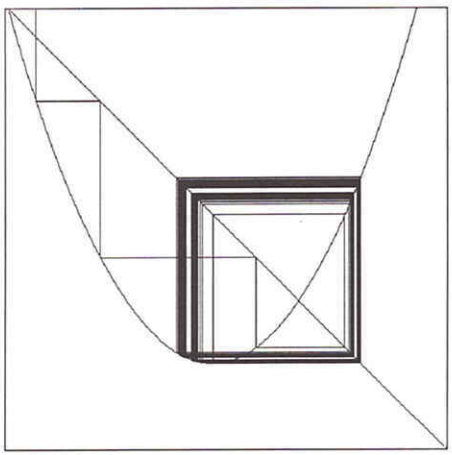
2-9 Grafische Iteration (Ausdruck des Programms feigenbaum5.c)



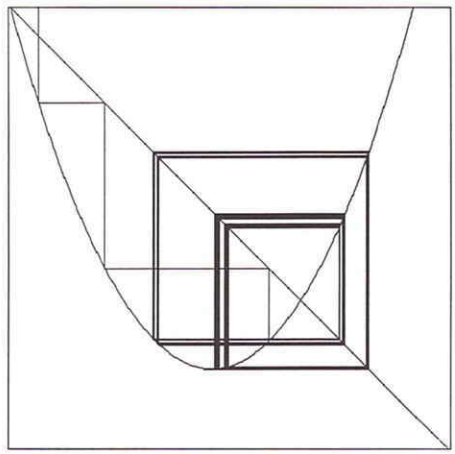
Ein Grenzwert



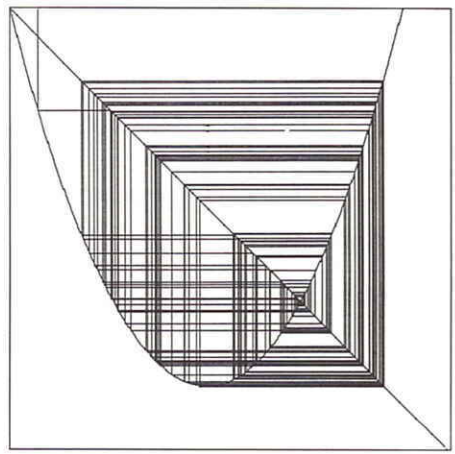
Zwei Grenzwerte



Vier Grenzwerte



Acht Grenzwerte



Chaos

Abbildung 2-9 (1) zeigt für die Startwerte $p_{\emptyset}$ = .1 und k = 2.Ø die Konvergenz gegen einen Grenzwert (=1). Für dieses Bild wurden 1000 Iterationen durchlaufen. Das Ergebnis ist für uns mittlerweile nicht mehr erstaunlich, denn vom Feigenbaum wissen wir bereits, daß es für diese Startwerte nur einen Grenzwert gibt. Interessanter wird es schon in Abbildung 2-9 (2) ($p_{\emptyset}$ = .1, k = 2.4 und n_{max} = 1ØØ); wir erkennen zwei Grenzwerte. Bei Abbildung 2-9 (3) ($p_{\emptyset}$ = .1, k = 2.4456 und n_{max} = 1ØØ) und Abbildung 2-9 (4) ($p_{\emptyset}$ = .1, k = 2.55, n_{max} = 1ØØ) hat sich die Anzahl der Grenzwerte jeweils wie zu erwarten war verdoppelt und für einen Startwert aus dem Chaosbereich (Abbildung 2-9 (5) – $p_{\emptyset}$ = .1, k = 2.8 und n_{max} = 1ØØ) füllt sich das Bild langsam völlig mit Linien. Aber selbst hier erkennt man Zonen in denen vermehrt Linien auftreten. Diese Häufung von Werten ist auch beim Feigenbaum in der Ausprägung stärkerer Linien zu sehen.

Ich denke, diese Auswertung trägt sehr zum Verständnis bei und sollte nicht unterschätzt werden. Gerade dazu noch eine Anmerkung: Während ich dieses Buch geschrieben habe, führte ich meine Ergebnisse natürlich meinen Freunden und Bekannten vor. Die Auswertungen der Verhulst-Gleichung stießen auf das meiste Desinteresse, denn die Bilder der Auswertungen komplexer Systeme sind bei weitem ausdrucksvoller. Ich finde aber gerade, um ein gewisses Grundverständnis zu erreichen ist es sinnvoll, diesen Weg zu gehen und erst einmal zu versuchen, dieses einfache System zu verstehen. Wie wir sehen werden, sind viele Feststellungen übertragbar! Jeder der sich genauer mit der Chaosforschung beschäftigen will, sollte dieses Kapitel nicht überblättern.

Damit jeder diese Auswertungsmethode selbst untersuchen kann, hier das entsprechende Programm:

Programm: *feigenbaum5.c*

```
/*********************************************************************/
/* feigenbaum41.c                                        Ø7.Ø1.88 */
/* Version 1.Ø                                                    */
/*                                                                */
/*                                                                */
/* Auswertung der Folge p(n+1)=p(n)*(1+k*(1-p(n))) durch Auftragen */
/* von p(n) nach Rechts und p(n+1) nach Unten fuer fest gewaehl-*/
/* tes k und Vergleichsparabel, um die mit Teilungspunkte verstaend- */
/* lich zu machen. (Eingabemoeglichkeit der Parameter aus dem CLI.) */
/*                                                                */
/* fuer Markt+Technik geschrieben von Heiko Knappe                */
/*                                                                */
/* Komplexe Dynamische Systeme                                    */
/* Feigenbaeume - ein Wachstumsproblem                            */
/*********************************************************************/


#include <exec/types.h>
#include <intuition/intuition.h>

#define WIDTH  64Ø        /* Screen Modus */
#define HEIGHT 5ØØ
#define DEPTH  2
#define FLAGS  HIRES+LACE

#define XVAL 1.5 /* Wertebereich */
#define YVAL 1.5
#define XWIDTH 5ØØ
#define YHEIGHT 5ØØ


struct IntuitionBase *IntuitionBase;
struct GfxBase *GfxBase;
struct Screen *MyScreen;
struct RastPort *rp;
struct NewScreen MyNewScreen=
 {Ø,Ø,WIDTH,HEIGHT,DEPTH,1,Ø,FLAGS,CUSTOMSCREEN,Ø,Ø,Ø,Ø};

float pØ=.1,pneu,palt;     /* Start-Population / Population    */
float k=2;                 /* rel. Wachstums-Rate              */
int n,nmax=1Ø;             /* Anzahl der Iterationen           */


/**********************/
/* Hauptprogramm      */
/**********************/

main(argc,argv)
int argc;
char *argv[];
{
 if((argc!=2)&&(argc!=1))
 {
    printf("Eingabe: %s <nmax,pØ,k>\n",argv[Ø]);
    printf(" nmax : Anzahl der Iterationen\n");
    printf(" pØ   : Anfangswert\n");
    printf(" k    : Wachstumsrate\n");
    close_things();
 }
```

```
60   if(argc==2)
61      if(sscanf(argv[1],"%d,%f,%f",&nmax,&pØ,&k)!=3)
62      {
63         printf("falsche Eingabe\n");
64         close_things();
65      }
66   printf("Parameter:\n nmax=%d\n pØ=%f\n k=%f\n",nmax,pØ,k);
67   open_things();
68   parabel_(); rech();
69   while(MyScreen->TopEdge<1);
70   close_things();
71  }
72
73
74  /***********************/
75  /* Initialisierungs-   */
76  /* routinen            */
77  /***********************/
78
79  open_things()
80  {
81   IntuitionBase=OpenLibrary("intuition.library",Ø);
82   if(IntuitionBase==Ø)
83      close_things();
84   GfxBase=OpenLibrary("graphics.library",Ø);
85   if(GfxBase==Ø)
86      close_things();
87   MyScreen=OpenScreen(&MyNewScreen);
88   if(MyScreen==Ø)
89      close_things();
90   rp=&(MyScreen->RastPort);
91   init_colors();
92  }
93
94  close_things()
95  {
96   if(IntuitionBase!=Ø) CloseLibrary(IntuitionBase);
97   if(GfxBase!=Ø) CloseLibrary(GfxBase);
98   if(MyScreen!=Ø) CloseScreen(MyScreen);
99   exit();
100 }
```

```
101
102  init_colors()
103  {
104   SetRGB4(&(MyScreen->ViewPort),Ø,Ø,Ø,Ø);
105   SetRGB4(&(MyScreen->ViewPort),1,15,Ø,15);
106   SetAPen(rp,1);
107  }
108
109
110  /***********************/
111  /* Berechnung          */
112  /***********************/
113
114  parabel_()
115  /* zeichnet die Vergleichsparabel und die Winkelhalbierende */
116  {
117   short x,y;
118
119   Move(rp,Ø,Ø); /* Parabel zeichnen */
120   for(x=Ø;x<XWIDTH;++x,palt=x*(float)XVAL/XWIDTH)
121   {
122      pneu=-k*palt*palt+(k+1)*palt;
123      y=pneu/YVAL*YHEIGHT;
124      if(y<Ø) y=Ø;
125      if(y>YHEIGHT) y=YHEIGHT;
126      Draw(rp,x,y);
127   }
128   Move(rp,Ø,Ø); /* Winkelhalbierende zeichnen */
129   Draw(rp,XWIDTH,YHEIGHT);
130   Move(rp,Ø,YHEIGHT); /* Achsen einzeichnen */
131   Draw(rp,Ø,Ø); Draw(rp,XWIDTH,Ø);
132  }
133
134  rech()
135  {
136   short x,y;
137
138   palt=pØ;
139   x=palt/XVAL*XWIDTH;
140   if(x<Ø) x=Ø;
141   if(x>XWIDTH) x=XWIDTH;
142   Move(rp,x,Ø);
```

```
143  for(y=x,n=Ø;n<nmax;++n)
144  {
145     Draw(rp,y,y);
146     pneu=palt*(1+k*(1-palt));
147     x=palt/XVAL*XWIDTH;
148     y=pneu/YVAL*YHEIGHT;
149     if(x<Ø) x=Ø;
15Ø     if(x>XWIDTH) x=XWIDTH;
151     if(y>YHEIGHT) y=YHEIGHT;
152     if(y<Ø) y=Ø;
153     Draw(rp,x,y);
154     palt=pneu;
155  }
156 }
```

Zeigt eigentlich nur die Feigenbaum-Formel ein chaotisches Verhalten, oder gibt es auch andere Folgen die ähnliche Eigenschaften aufweisen? Wie ändern sich die Bilder?

Wir sind zum Beispiel nicht gezwungen, wie bisher die Folge nur vom vorherigen Wert abhängig zu machen, es ist durchaus sinnvoll, auch weitere Vorgängerwerte mitzuverwenden. Dies geschieht zum Beispiel beim Verhulst-Attraktor, der durch die Formel

$$p_{n+1} = p_n + .5*k*(3*p_n*(1-p_n) - p_{n-1}*(1 - p_{n-1}))$$

beschrieben wird. Diese Gleichung ist aus der Mathematik als Näherungsverfahren zur Lösung einer Differentialgleichung (DGL) bekannt. Eine DGL ist eine Gleichung, bei der eine Funktion von ihren Ableitungen abhängt. Ein einfacher Fall ist zum Beispiel die Gleichung `f'(x)=f(x)`.

Es gibt nun Verfahren, eine solche DGL zu lösen. In diesem Falle ist die Funktion `f(x)=exp(x)` eine Lösung, denn ihre Ableitung ist ebenfalls `exp(x)`, erfüllt also diese Gleichung. Für kompliziertere Aufgabenstellungen ist es jedoch oft nicht möglich, mit vertretbarem Aufwand noch eine Funktion zu finden (sofern überhaupt möglich!), die eine Lösung darstellt. Wie bei der Nullstellensuche (siehe nachfolgendes Kapitel) greift man deshalb gerne zu numerischen Verfahren, die heute leicht mit einem Rechner durchgeführt werden können.

Als einfachstes Beispiel wollen wir das sogenannte Euler-Verfahren verwenden:

$$y_{n+1} = y_n + k*y_n' \ (+ f) \qquad (1)$$

(f ist der gemachte Fehler – er ist proportional zu k^2.)

Mit diesem Verfahren versuchen wir für die DGL

$$y' = y * (1-y) \qquad (2)$$
$$(\text{bzw. } y_n' = y_n * (1-y_n))$$

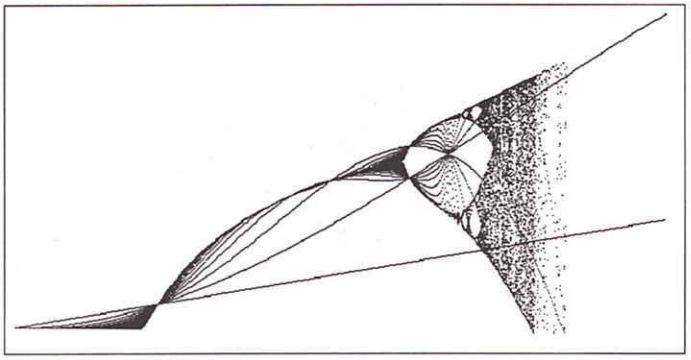
iterativ eine Lösung zu finden. Setzen wir Gleichung (2) in (1) ein, so erhalten wir:

$$y_{n+1} = y_n + k * y_n * (1-y_n)$$

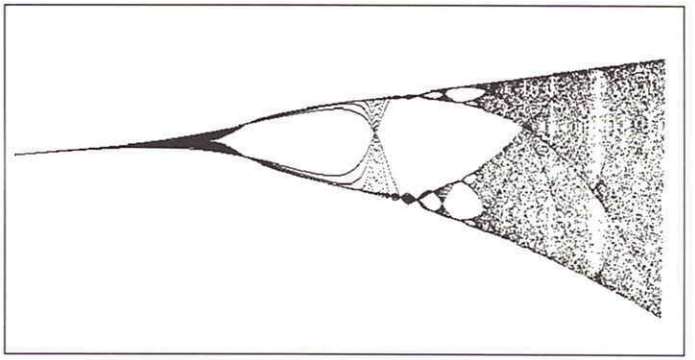
Diese Formel ist uns bereits bekannt! Sie beschreibt unseren alten Bekannten, das Feigenbaum-Diagramm. Man darf sich aber nicht dazu hinreißen lassen, das Diagramm unmittelbar als Lösung der DGL (2) zu sehen! Eine Lösung erhalten wir, wenn wir uns die Randbedingungen der DGL vorgeben, zum Beispiel `p = .1` und `k = 1`, und diese Werte in die Formel einsetzen. Nacheinander erhalten wir jetzt die Folgenwerte y_n. Verbinden wir alle Folgenwerte miteinander, so erhalten wir eine Kurve, die Lösung der Differentialgleichung. Wer also meint, wir hätten uns völlig von der praktischen Anwendung entfernt, wird nun hoffentlich einsehen, daß es nicht so ist!

Zum Schluß dieses Kapitels nochmals ein Tip: Experimentieren Sie selbst! Verwenden

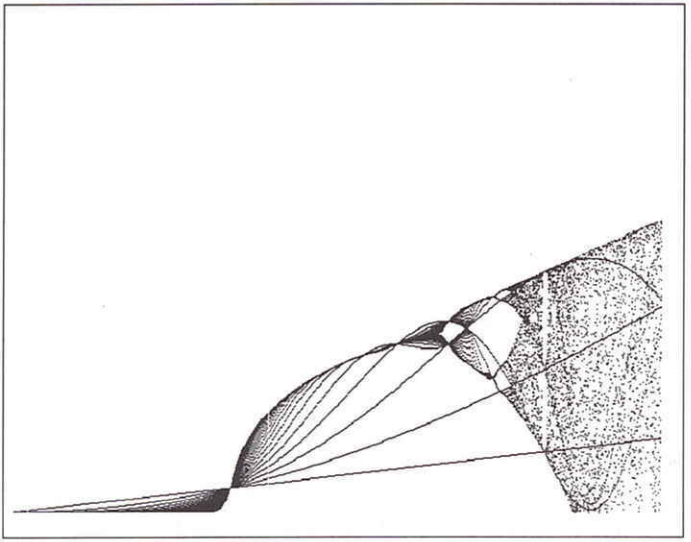
2-10 Feigenbaum Diagramme mit anderen Folgen



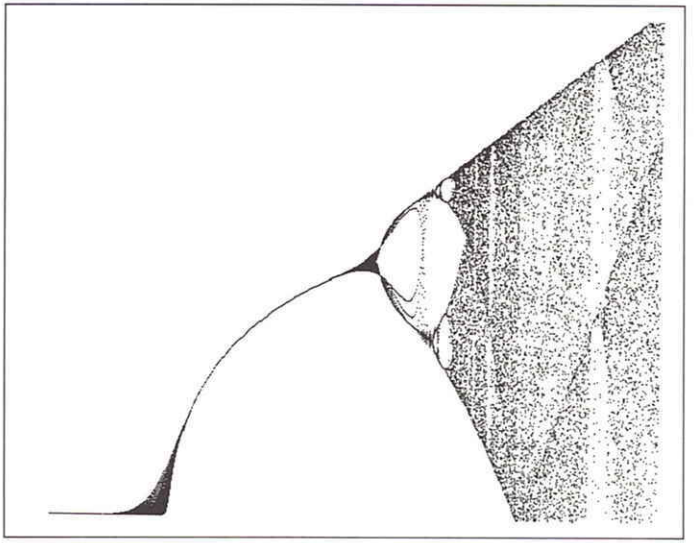
$p_{n+1} = k*p_n*(1-p_n)$



$p_{n+1} = k*p_n*(1-p_n)$



$p_{n+1} = k*p_n*(1-p_n^2)$



$p_{n+1} = k*sin(p_n)*cos(p_n)$

Sie zum Beispiel andere Funktionen, oder, wie oben angesprochen, eine Folge, deren Glieder von zwei Vorgängerwerten bestimmt wird. Zum Beispiel die Feigenbaum-Diagramme für folgende Funktionen:

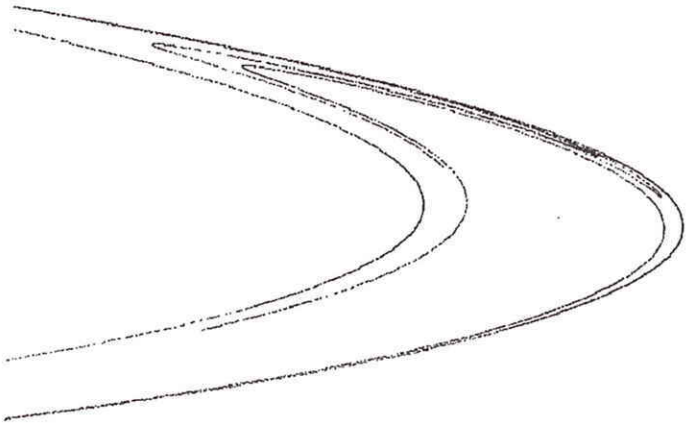
$p_{n+1} = k*p_n*(1-p_n)$ (Abbildung 2-10 (1)/(2))
$p_{n+1} = k*p_n*(1-p_n^2)$ (Abbildung 2-10 (3))
$p_{n+1} = k*\sin(p_n)*\cos(p_n)$
(Abbildung 2-10 (4))

Die Ergebnisse weisen trotz völlig unterschiedlicher Funktionen ähnliche Ergebnisse auf!

Oder versuchen Sie es doch einmal mit einer Funktion der Art:

$x_{n+1} = y_n - a*x_n^2+1$
$y_{n+1} = b*x_n$

Wobei y_n nach oben und x_n nach rechts aufgetragen wird. Es entsteht eine Figur, die Henon-Attraktor genannt wird. Für Abbildung 2-11 wurden folgende Startwerte gewählt: $x_\emptyset = y_\emptyset = \emptyset$ und `a = 1.3` bzw. `b = .3`, 1000 Iterationen wurden gezeichnet. Das Bild entsteht durch Verbinden der aufeinanderfolgenden Ergebniswerte durch eine Gerade. Deutlich erkennt man die einer Parabel ähnliche Figur, die analog zu der Auswertung in dem Programm *feigenbaum4.c* entsteht.

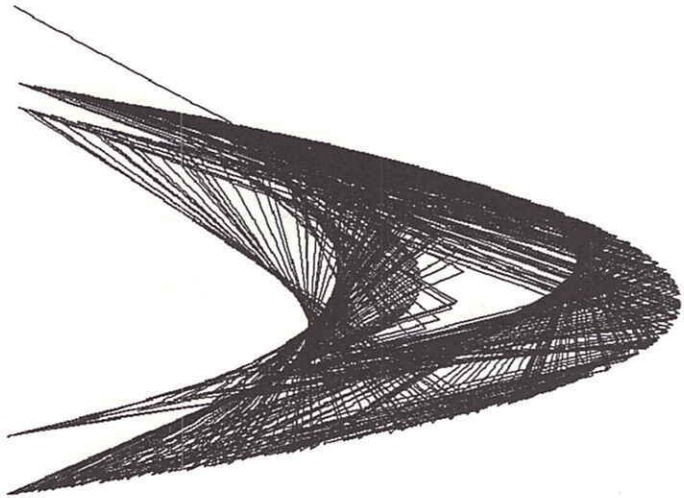


$$x_{n+1} = y_n - a*x_n^2 + 1$$
$$y_{n+1} = b*x_n$$

2-11-1 Der Henon-Attraktor

Wie bereits angesprochen, werden wir die Eigenschaften dieser »trivialen« auch bei den nun folgenden komplexeren Systemen wiederfinden. Deshalb zum Schluß eine Zusammenfassung der wichtigsten Erkenntnisse:

- Wichtigste Grundlage für ein dynamisches System ist die Rückkopplung, zum Beispiel die Verwendung einer rekursiven Folge.



2-11-2 Der Henon-Attraktor einmal anders (aufeinanderfolgende Punkte werden verbunden)

- Die Art der Auswertung ist von großer Bedeutung, um das Verhalten des Systems besser verstehen zu können.
- Das Verhalten der Systeme ist in manchen Gebieten nicht vorhersagbar. In diesen führt aber eine wiederholte Berechnung mit den gleichen Startwerten ganz sicher zum gleichen Ergebnis! Es handelt sich also nicht um Zufallserscheinungen.
- Wie kompliziert sich ein Attraktor (das heißt die »Grenzfigur«, gegen die die Folge strebt) gestaltet, hängt von der Zahl der Eingangsparameter ab.
- Für die Untersuchung besonders interessant sind natürlich die Grenzgebiete zwischen dem Konvergenz- und Divergenzbereich.

Um die folgenden Systeme zu verstehen ist die Kenntnis der komplexen Zahlen unbedingt notwendig. Bevor wir richtig loslegen können, müssen wir uns noch mit ein wenig Mathematik beschäftigen.

3 Mathematische Grundlagen

3.1 Was sind komplexe Zahlen?

Bei dem Wort »komplex« wird vielen Menschen unwohl, und bei dem Gedanken, mit imaginären Zahlen zu rechnen, schütteln sie nur uninteressiert den Kopf, denn sie meinen, imaginäre Zahlen seien nur die Auswüchse der Mathematik und eigentlich praktisch nicht anwendbar. Die Schüler, die in der Schule diese unwirklichen Zahlen erstmals kennenlernen, reagieren auf den Stoff meist mit reichlichem Unverständnis und Desinteresse, welches dann oft sehr lange anhält. Aber man sollte sich nicht gleich zu Anfang von dem Wort komplex abschrecken lassen. Nach einigem Beschäftigen mit den scheinbar unwirklichen, nicht greifbaren Zahlen wird verständlich, daß die komplexen Zahlen an der Einfachheit ihrer Definition, der Anwendung und der Anschaulichkeit den reellen Zahlen in keiner Weise nachstehen.

Aber warum sind die Mathematiker überhaupt auf den Gedanken gekommen, mit solchen Zahlen zu rechnen? Stellen wir uns doch einmal folgendes Problem vor: Sowohl das Produkt als auch die Summe zweier Zahlen soll gleich zwei sein, also

$(a+b)=2$ und $(a*b)=2$.

Löst man die erste Gleichung nach a auf und setzt dieses Ergebnis in die zweite ein, so erhält man eine Bestimmungsgleichung für die Zahl b. In unserem Fall:

$a=(2-b)$
$\Rightarrow (2-b)*b=2$
$\Rightarrow b^2-(2*b)+2=\emptyset$

Diese quadratische Gleichung kann nun mit Hilfe der quadratischen Ergänzung gelöst werden:

$(b-1)^2-1+2=\emptyset$
$\Rightarrow (b-1)^2+1=\emptyset$
$\Rightarrow (b-1)^2=(-1)$

Rechnet man mit reellen Zahlen, so muß die Lösung an dieser Stelle abgebrochen werden, denn im Bereich der reellen Zahlen ($\mathbb{R}$) gibt es keine Wurzeln aus negativen Zahlen. Machen wir jetzt doch einmal ein Gedankenexperiment: Wir gehen davon aus, daß es die Wurzel aus (-1) gibt und setzen sie gleich einem Parameter i, den wir willkürlich annehmen. Diese Zahl i ist nun keine reelle Zahl mehr, sondern eine neue Art von Zahl die wir uns gerade ausgedacht haben. Sie ist also sozusagen imaginär. Das Quadrat dieser Zahl ist gleich -1, also $i^2=(-1)$.

Jetzt schreibt sich die letzte Gleichung $(b-1) = i$ und damit ergibt sich:

$b=(1+i)$ und $a=(1-i)$.

Daß dies Ergebnis sinnvoll ist, zeigt eine Probe:

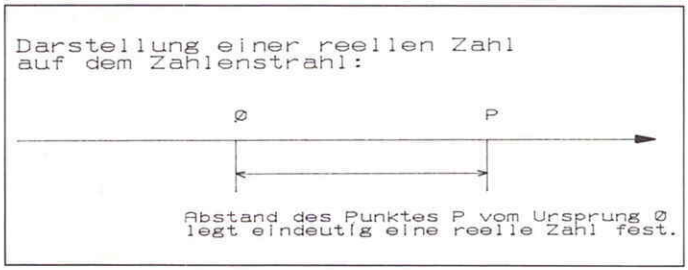
$a*b=(1+i)*(1-i)=1-i^2+i-i=1-(-1)=2$ und
$a+b=(1+i)+(1-i)=2$.

Diese von uns neu eingeführte Zahlenart macht es also möglich, diese vorher nicht lösbare Aufgabe sinnvoll zu beenden. Mit Hilfe der reellen Zahlen wären wir dazu nicht in der Lage gewesen. Dieser Gedanke war der Auslöser für die Einführung der komplexen Zahlen. Der Einsteiger stellt hier natürlich die Frage nach dem Sinn einer solchen Erweiterung, denn bisher bleibt das oben berechnete Ergebnis ohne besonderen praktischen Nutzen. Wie sich nach konsequentem Weiterdenken ergibt, stellen die komplexen Zahlen eine Obermenge der reellen Zahlen dar. Beim Rechnen gelten die gleichen Rechenregeln und Rechengesetze. Dies allein mag immer noch nicht zufriedenstellen, aber es hat sich schnell gezeigt, daß mit Hilfe der komplexen Zahlen bereits nach wenigen Schritten Ergebnisse vorliegen, die mit reellen Zahlen erst durch sehr schwierige Umformungen zu erreichen sind. Heute sind die komplexen Zahlen aus vielen Bereichen der Mathematik und vor allem der Technik nicht mehr wegzudenken. An einer praktischen Bedeutung kann man also kaum zweifeln.

3.2 Neue Rechenregeln?

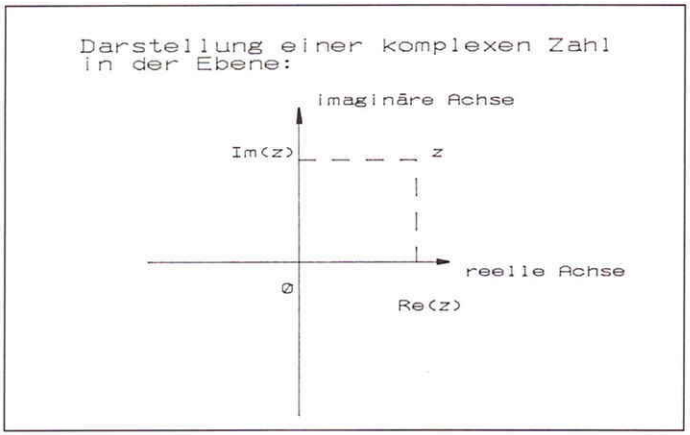
Wie bereits angesprochen, kann man mit komplexen Zahlen genauso rechnen wie mit den reellen, wenn man sich die Zahl `i` als Variable vorstellt. Diese Variable ist sozusagen ein Chamäleon, denn sie kann zu einer reellen Zahl werden, indem sie quadriert wird. Wenn auch das Rechnen grundsätzlich gleich bleibt, ist es dennoch notwendig, einige Eigenschaften der »neuen« Zahlen anzusprechen. Ohne ein wenig Mathematik geht es einfach nicht.

Für komplexe Zahlen gibt es verschiedene Darstellungsarten. Die erste ist die schon bereits im obigen Gedankenexperiment kennengelernte Darstellungsweise. Eine komplexe Zahl besteht demnach aus zwei Teilen, einem reellen und einem imaginären, der durch die Variable `i` angedeutet wird. Man spricht von einem sogenannten Real- und einem Imaginärteil der komplexen Zahl. Ein Beispiel macht dies verständlich: Die komplexe Zahl `(1+i)` besitzt den Realteil `1` und den Imaginärteil `1`. Man schreibt `Im(1+i)=1` für den Imaginär- und `Re(1+i)=1` für den Realteil, also besitzt zum Beispiel die komplexe Zahl `(3-4i)=(3-4*i)=z` den Realteil `Re(z)=3` und den Imaginärteil `Im(z)=(-4)`.



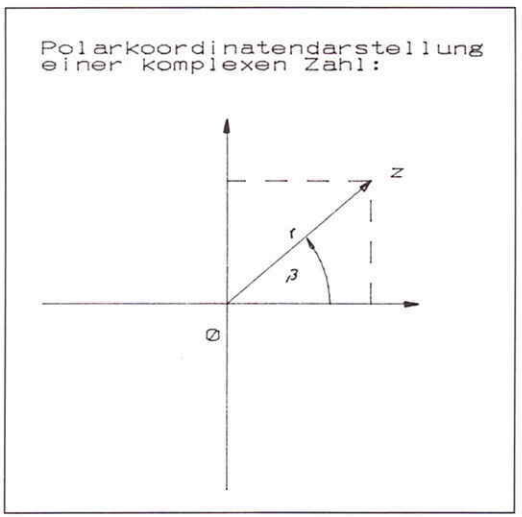
3-1

Eine zweite Darstellungsweise ist ebenfalls von großer Bedeutung, die sogenannte Polarkoordinatendarstellung. Um diese zu verstehen, verlegen wir uns zuerst auf die anschauliche Vorstellung von komplexen Zahlen als Vektoren. Die Veranschaulichung einer reellen Zahl war recht einfach. Auf einer Geraden läßt sich jedem Punkt genau eine reelle Zahl zuordnen (siehe Abbildung 3-1). Wie kann man sich nun aber eine komplexe Zahl vorstellen? Sie besteht aus zwei Teilen, die beide für sich reell sind. Es liegt also nahe, zwei Zahlengeraden, die die beiden reellen Zahlen vertreten sind, aufeinanderzustellen und sich jeden Punkt der dabei entstehenden Ebene als eine komplexe Zahl vorzustellen (siehe Abbildung 3-2). Geht man nun noch weiter und überlegt sich, wie man den Punkt außer mit Hilfe der Zerlegung in Real- und Imaginärteil charakterisieren kann, so liegt



3-2

die Verwendung der Polarkoordinaten nahe. Jede komplexe Zahl läßt sich damit durch eine Zahl und einen Winkel darstellen. Die Zahl gibt dabei den Abstand des Punktes vom Ursprung an (man spricht daher auch vom Betrag der Zahl) und der Winkel gibt an, wie weit die x-Achse gegen den Uhrzeigersinn gedreht werden muß, um sich mit der Verbindungsgeraden Ursprung – Punkt zu decken. Bei dem Winkel spricht man auch von dem Argument oder Phase der komplexen Zahl und schreibt ß=arg(z) (siehe Abbildung 3-3).



3-3

Die beiden Darstellungsformen lassen sich natürlich ineinander überführen und besitzen die gleiche Aussagekraft. Ob man eine komplexe Zahl z=a+b*i oder z=(r,ß) (r Betrag, ß Winkel) schreibt, ist also egal. So gilt für eine komplexe Zahl z=(a+b*i), also mit Re(z)=a und Im(z)=b (a und b reelle Zahlen):

$r=\sqrt{(a^2+b^2)}$ und

$$ß = \arctan\left(\frac{b}{a}\right) = \arctan\left(\frac{Im(z)}{Re(z)}\right)$$

Bei der Berechnung von ß muß natürlich auf die Nichteindeutigkeit des Tangens geachtet werden.
Das umgekehrte Vorgehen ist auch nicht schwerer: (Eine komplexe Zahl mit Betrag r und Winkel ß ist gegeben.)

a=r*cos(ß) und b=r*sin(ß).

Diese Formeln folgen unmittelbar aus der geometrischen Veranschaulichung der komplexen Zahl in der Ebene. Die Zahl 1+i kann also auch mit dem Betrag $r = \sqrt{1+1} = \sqrt{2}$ und dem Winkel ß = arctan(1/1) = arctan(1) = 45° beschrieben werden. Umgekehrt kann man z = (4,3Ø°) als Re(z) = 4*cos(3Ø°) = 4*.5*$\sqrt{3}$ = 2*$\sqrt{3}$ und Im(z) = 4*sin(3Ø°) = 4*.5 = 2, genauso als z = 2*$\sqrt{3}$ + 2*i schreiben.

Die allgemeinen Rechenregeln

Addition und Subtraktion sind einfach auszuführen. Addiert man zwei komplexe Zahlen, so führt man die Berechnung für Real- und Imaginärteil getrennt aus. Definieren wir zum Beispiel z_1 = 1+i und z_2=3+i, so gilt:

$z_1+z_2=(1+i)+(3+i)=(1+3)+(1+1)*i=4+2*i$
$z_1-z_2=(1+i)-(3+i)=(1-3)+(1-1)*i=(-2)$

Die Multiplikation bereitet auch keine Schwierigkeiten, wenn man i immer als

Variable betrachtet. So berechnet man für $z_1 * z_2$ zum Beispiel:

$z_1*z_2=(1+i)*(3+i)=(1*3+1*i+i*3+i^2)=$
$(3-1)+i*(1+3)=2+4*i$

Bei der Divison wird es allerdings etwas schwieriger. Denn steht eine komplexe Zahl im Nenner des Bruches, so kann dieser Bruch nicht mehr unmittelbar in Real- und Imaginärteil aufgeteilt werden. Abhilfe schafft hier ein Kniff. Multiplizieren wir doch einmal $(2+i)$ und $(2-i)$ so erhalten wir:

$(2+i)*(2-i)=(2*2-2*i+2*i-i^2)=5$

Wie man sieht ist das erhaltene Ergebnis nun reell. Dies kann man mit jeder komplexen Zahl durchführen, indem man sie mit ihrer konjugiert komplexen multipliziert. Die konjugiert komplexe erhält man, indem man das Vorzeichen des Imaginärteils umkehrt. Die Zahl $5-3i$ besitzt also die konjugiert komplexe $5+3i$. Verwenden wir nun diese Erkenntnis, um den Nenner einer komplexen Zahl reell zu machen, so müssen wir also nur den Bruch mit der konjugiert komplexen des Nenners erweitern. Dies bedeutet für die Division von z_1 und z_2:

$$\frac{z_1}{z_2} = \frac{(1+i)}{(3+i)} = \frac{(1+i)\ *\ (3-i)}{(3+i)\ *\ (3-i)}$$

$$= \frac{3-i+3i-i^2}{9-3i+3i-i^2} = \frac{(4+2i)}{10}$$

Damit haben wir unser Ziel erreicht, der Bruch ist wieder in Real- und Imaginärteil darstellbar.

Auch wenn die Polarkoordinatendarstellung genau dieselbe Aussagekraft wie die Darstellung in Real- und Imaginärteil hat, so ist es uns bis jetzt noch nicht möglich, mit ihr zu rechnen. Dies wird erst mit der Eulerschen Gleichung möglich.

Entwicklt man die Exponentialfunktion $\exp(x)$ in einer Reihe, so erhält man folgendes Ergebnis:

$$\exp(x) = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots$$

Setzen wir nun für x eine imaginäre Zahl ein, so erhalten wir nach Trennung der erhaltenen Reihe in Real- und Imaginärteil:

$$\exp(iy) = \left(1 - \frac{y^2}{2!} \ldots\right) + i*\left(y - \frac{y^3}{3!} + \ldots\right)$$

Die erste Klammer ist nun die bekannte Reihe des $\cos(y)$ und die zweite die des $\sin(y)$. Es ist also $\exp(iy)=\cos(y)+i*\sin(y)$. Verwendet man nun diese Erkenntnis, so kann eine komplexe Zahl mit den Polarkoordinaten r und β auch $z = r * \exp(i*\beta)$ schreiben. Der Vorteil dieser Darstellung liegt vor allem bei der Multiplikation und Division.

Abschließend noch zwei Beispiele, die den Nutzen der komplexen Zahlen verdeutlichen:

1) Herleitung der Additionstheoreme

Multipliziert man zwei komplexe Zahlen $z_1=\exp(ix)$ und $z_2=\exp(iy)$, so erhält man:

$z_1*z_2=\exp(ix)*\exp(iy)=$
$\exp(i*(x+y))=$
$\cos(x+y)+i*\sin(x+y)$ und
$z_1*z_2=$
$(\cos(x)+i*\sin(x))*(\cos(y)+i*\sin(y))$
$=\cos(x)*\cos(y)-\sin(x)*\sin(y)+$
$i*(\cos(x)*\sin(y)+\sin(x)*\cos(y))$

Da beide Ergebnisse in Real- und Imaginärteil übereinstimmen müssen, gilt:

$\cos(x+y)=\cos(x)*\cos(y)+\sin(x)*\sin(y)$
und
$\sin(x+y)=\sin(x)*\cos(y)+\cos(x)*\sin(y)$

Dies sind die sogenannten Additionstheoreme. Versucht man sie geometrisch herzulei-

ten, so ist dies nur mit erheblichem Mehraufwand möglich.

2) Als zweites Beispiel die Moivre-Formeln

$\exp(2*(ix))=\exp(ix)^2=(\cos(x)+i*\sin(x))^2$
$=\cos(x)^2-\sin(x)^2+2*i*\cos(x)*\sin(x)$

oder

$\exp(2*(ix))=\cos(2*x)+i*\sin(2*x)$

Vergleicht man nun beide Ergebnisse, so erhält man

$\cos(2*x)=\cos(x)^2-\sin(x)^2$ und
$\sin(2*x)=2*\sin(x)*\cos(x)$.

Genau das gleiche Vorgehen ist natürlich auch für 3,4,5,... möglich.

3.3 Rekursive Folgen

Wie wir bereits bei den Feigenbäumen gesehen haben, ist Rückkopplung die Grundlage für das Entstehen von Chaos und von inneren Strukturen. Sie ist sozusagen das Lebenselixier fraktaler Grafiken. Aber was wird denn nun eigentlich rückgekoppelt und wie werden die Ergebnisse interpretiert, um die faszinierenden Bilder zu erzeugen?

Auch hier kommen wir ohne ein wenig Mathematik nicht aus. Wichtigste Grundlage sind die sogenannten Folgen. Eine Folge ist einfach eine Abbildungsvorschrift, deren Definitionsbereich im allgemeinen die natürlichen Zahlen sind. Sie liefert in Abhängigkeit der natürlichen Zahl eine Folge von Zahlen. Deshalb auch der Begriff Zahlenfolge. Zum Beispiel liefert $a_n=\frac{1}{n}$ mit $n=1,2,3,4,\ldots$ die Zahlenfolge $a_n=\frac{1}{2},\frac{1}{3},\frac{1}{4}\ldots$

Wie man erkennt, werden die Glieder (d.h. die Ergebnisse der einzelnen Abbildungsvorschriften) mit wachsendem n immer kleiner und für einen beliebig großes n fast Null. Dieses Verhalten der Folge nennt man konvergent gegen den Wert Null. Konvergenz bedeutet also die Existenz eines Grenzwertes (in unserem Fall 0), der für n gegen Unendlich angenommen wird. Eine Folge die nicht konvergent ist, bezeichnet man als divergent und unterscheidet hier noch zwischen bestimmt und unbestimmt divergent.

$a_n=(-1)^n \qquad n=\emptyset,1,2,3,4,\ldots$

ist zum Beispiel eine unbestimmt divergente Folge. Ihre Werte schwanken abwechselnd sprungartig zwischen positiven (1) und negativen (-1) Werten. Eine bestimmt divergente Folge geht dagegen für n gegen Unendlich selbst gegen Unendlich.

Auch hier ein Beispiel:

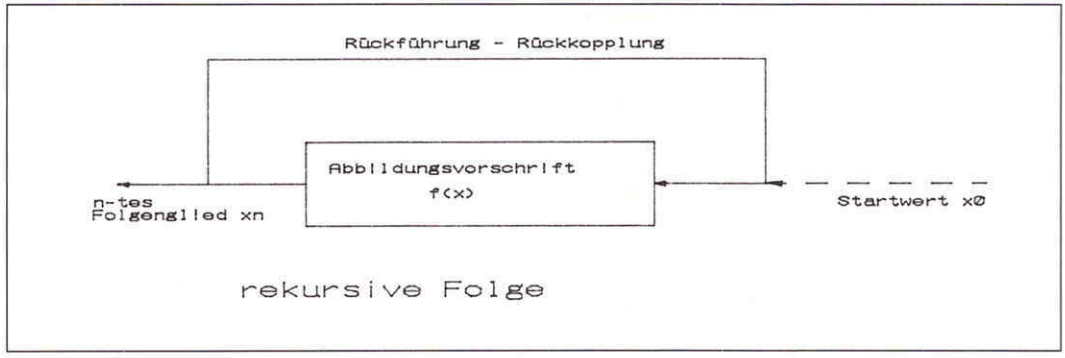
$a_n=n \qquad n=\emptyset,1,2,3,4,\ldots$

Diese Folge wächst bei steigendem n streng monoton und wird für n gegen Unendlich ebenfalls sehr groß werden.

Das Rechnen mit Folgen, Grenzwerten und dann mit Reihen erfordert eine ganze Menge Mathematik und ist zum Teil nicht mehr trivial. Glücklicherweise brauchen wir diese mathematischen Grundlagen zur Erzeugung schöner Bilder in erster Linie nicht. Sie deshalb hier noch weiter auszuführen hat also nicht sehr viel Sinn. Der Interessierte findet sicher ausreichend Informationen in jedem Mathematikbuch.

Wie bereits oben angesprochen, geht es bei komplexen dynamischen Systemen oder einfachen fraktalen Grafiken nicht ohne Rückkopplungen der verschiedensten Art. Im Falle der Apfelmännchen, den Julia-Mengen oder den Auswertungen nach dem Newton-

3-4



Verfahren und ihren Abwandlungen finden diese Rückkopplungen durch rekursive Folgen statt.

Was eine Folge ist, wissen wir nun. Eine rekursive Folge erhalten wir, wenn wir das Ergebnis eines Folgenschrittes der Berechnung des nächsten Glieds zugrunde legen (siehe Abbildung 3-4). Zu allererst müssen wir also von dem Definitionsbereich der ganzen Zahlen zu dem der reellen Zahlen übergehen. Haben wir also zum Beispiel die Abbildungvorschrift bzw. Funktion x^2, so erhalten wir daraus die rekursive Folge

$$x_{n+1} = x_n^2 \quad (\text{mit } n=Ø,1,2,3,\ldots).$$

Es ist verständlich, daß diese Folge von dem Startwert $x_Ø$ abhängt, deshalb ihr Verhalten für viele Folgenglieder nicht leicht vorherzusagen ist. In unserem einfachen Beispiel können wir sicher sagen, daß für einen Startwert $x_Ø$ größer oder gleich 1 (bzw. kleiner oder gleich -1) die Folgenglieder monoton wachsen werden, die Folge also gegen Unendlich bestimmt divergent ist. Dagegen wird sie für Anfangswerte zwischen -1 und 1 gegen den Wert Null streben. Diese rekursive Folge hat also in Abhängigkeit von den Startwerten zwei Grenzwerte (diese sind die Attraktoren dieser Folge). Man kann sich auch vorstellen, ein Folgenglied nicht nur von seinem Vorgänger, sondern auch von dessen Vorgänger abhängig zu machen. Daß dann die Verhältnisse natürlich immer schwerer durchschaubar werden, dürfte jedem klar sein.

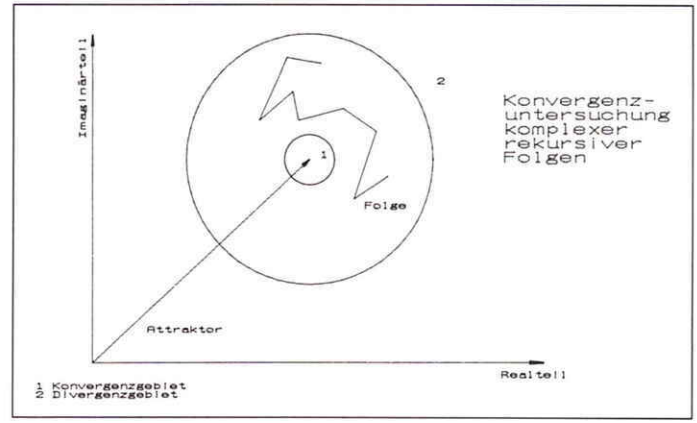
Die rekursive Folge des obigen Beispiels zeigt ein definiertes und nicht überraschendes Verhalten. Die Grenzen der Einzugsgebiete der beiden Attraktoren sind genau definiert und enthalten sicher keine Überraschungen. Außerdem arbeiten wir nur auf dem Zahlenstrahl. Wie schnell jedoch definiertes und übersichtliches Verhalten undurchsichtig oder sogar chaotisch werden kann, haben wir bereits bei den Feigenbaum-Diagrammen zu spüren bekommen. Auch hier hatten wir eine einfache Abbildungsvorschrift (p+k*p*(1-p)) zum Ausgangspunkt, und mußten bei eingehender Betrachtung feststellen, wie seltsam sich die Folge in ihrem Konvergenzbereich verhält und wie sehr es auf die Art der Auswertung ankommt. Wir mußten aber auch feststellen, daß sich bei erheblicher Änderung der Abbildungsfunktion (zum Beispiel k*sin(p)*cos(p)) das Aussehen des Attraktors, also des Feigenbaum-Diagramms, kaum änderte. Die Entdeckungsreise war also auch hier schnell zu Ende.

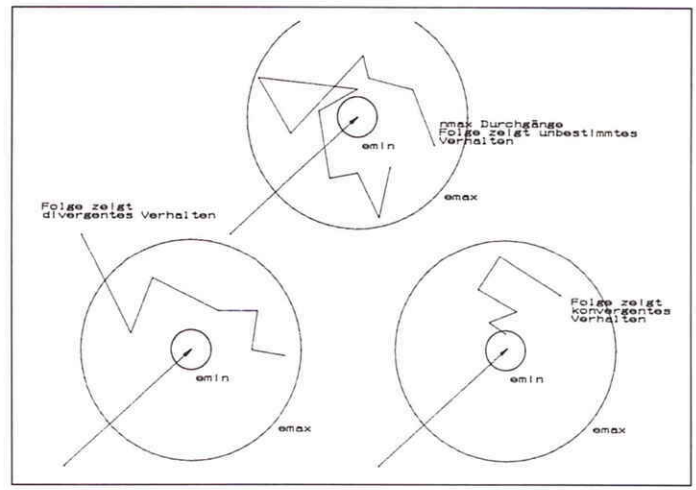
Wie jeder schon nach der vorigen Erklärung der komplexen Zahlen ahnen wird: Wer sagt

uns denn, daß der Wertebereich der Folgen aus reellen Zahlen bestehen muß, warum eigentlich nicht aus komplexen? Als Eingangswerte der Folgen haben wir nun also keine reellen Zahlen mehr, sondern komplexe. Wir gehen also vom Zahlenstrahl auf die Zahlenebene über. Wie schwierig, ja sogar unvorhersagbar das Folgenverhalten sein wird, zeigen die vielen Bilder aus dem Randbereich mehrerer Attraktoren.

Die Auswertung dieser Folgen hängt natürlich immer von den jeweiligen Untersuchungskriterien ab und wird an entsprechender Stelle angesprochen werden. Im allgemeinen (zum Beispiel Julia-Mengen und Apfelmännchen) wird folgendermaßen vorgegangen: Es wird ein beliebiger Punkt (sinnvollerweise ein Attraktor, also Grenzwert der Folge) gewählt. Der Abstand der einzelnen Folgenglieder von diesem Punkt wird nun herangezogen, um zu beurteilen, ob die Folge divergiert oder konvergiert. Am besten veranschaulichen wir uns das Vorgehen an Hand eines Bildes. Um den Attrakor $z_{\emptyset}$ zeichnen wir zwei Kreise mit dem Radius e_{min} und e_{max}. Eine sinnvolle Untersuchung kann natürlich ab jetzt nur noch für Startwerte der Folge aus dem Gebiet zwischen beiden Kreisen erfolgen. Außerdem sollte e_{min} klein (zum Beispiel Ø.ØØØ1) und e_{max} groß (zum Beispiel 1ØØØØ) gewählt werden. Jetzt tragen wir die einzelnen Folgenglieder, die natürlich auch komplex sind, in der Zahlenebene auf. Gelangt nun ein Wert in den innerern Kreis, so ist für uns die Folge für diesen Startwert konvergent, verläßt die Folge den äußeren Kreis, so divergiert sie. Ein dritter Fall ist natürlich ebenfalls noch möglich und muß beachtet werden: Ist entweder in dem Gebiet zwischen den beiden Kreisen ein weiterer Attraktor oder verläßt die Folge das Gebiet erst nach sehr vielen Durchläufen, so müssen wir um die Auswertungszeit in einem vernünftigen Rahmen zu halten oder gar einen Absturz zu verhindern, die Berechnung der Folgenglieder nach einer maximalen Anzahl von Durchläufen abbrechen (im weiteren n_{max}) (siehe Abbildung 3-5 (1)/(2)). Je höher

3-5 (1)





3-5 (2)

wir n_{max} wählen, desto länger wird die Berechnung des Bildes dauern, desto genauer erkennbar werden aber die Randstrukturen der Einzugsbereiche 0 (Folgenglied erreicht den Kreis um den Attraktor mit Radius e_{min}) und Unendlich (Folgenglied verläßt den Kreis

mit Radius e_{max}). Das Aussehen des Bildes hängt somit vor allem von den drei Parametern e_{min}, e_{max} und n_{max} (Abbruchkriterien) und natürlich der zugrunde liegenden Abbildungsfunktion ab. Man muß sich natürlich nicht auf die Konvergenzuntersuchung gegen einen Punkt beschränken. Analog zu den Untersuchungen des Newton-Verfahrens im folgenden Abschnitt ist es durchaus sinnvoll und zulässig, Konvergenz gegen mehrere Punkte zu interpretieren. Doch meist reicht es für uns nicht aus zu erkennen, ob eine Folge divergent, konvergent oder unbestimmt ist, sondern wir wollen die Dauer, das heißt die Anzahl der notwendigen Iterationen bis das Abbruchkriterium eingetroffen ist, ebenfalls auswerten. Die Anzahl der notwendigen Durchläufe interpretieren wir dann entweder als Farbe oder Höhe eines Gebirges.

4 Der Vorstoß ins Chaos

4.1 Die Folge z^2-c – die Mandelbrot-Menge, Julia-Mengen und ihre Verwandten

Im letzten Kapitel haben wir uns mit der Auswertung komplexer Folgen befaßt. Das Newton-Verfahren ist recht aufwendig, besonders bei der Nullstellensuche für Funktionen höheren Grades. Deshalb wollen wir uns zuerst mit einer wesentlich einfacheren nichtlinearen »quadratischen« Rückkopplung beschäftigen:

$z_{n+1}=z_n^2-c$

Das Verhalten dieser rekursiven Folge in der komplexen Zahlenebene hat der aus Polen stammende amerikanische Mathematiker Benoit B. Mandelbrot erstmals 1980 untersucht. Trotz der einfachen Formel erhalten wir faszinierende Bilder mit reichhaltigen Strukturierungen und den typischen Merkmalen fraktaler Grafiken, wie zum Beispiel der Selbstähnlichkeit.

Die Folgewerte von $z_{n+1} = z_n^2 - c$ werden sicher von dem Startwert $z_\emptyset$ und der Wahl des Parameters c abhängen. Beides sind komplexe Werte mit je einem Imaginär- und Realteil. Wir haben also vier Parameter von deren Wahl es abhängen wird, wie die Grafik aussieht.

Zur Berechnung ist folgendes zu sagen: Gehen wir davon aus, daß z_n und c die Form $z_n=re_n+im_n*i$ und $c=c_{re}+c_{im}*i$ haben, wobei re_n, im_n, c_{re} und c_{im} reellwertige Zahlen sind, so erhalten wir den Imaginär- und Realteil des nächsten Folgegliedes z_{n+1} als

$re_{n+1} = re_n^2 - im_n^2 - c_{re}$ und
$im_{n+1} = 2*re_n*im_n - c_{im}$.

Diese Formel wollen wir in den Bereichen Unendlich, Null und in ihrem zyklischem Verhalten untersuchen. Dazu gehen wir wie bei der komplexen Nullstellensuche vor. Wir legen folgende drei Abbruchkriterien der Berechnung zugrunde:

- Ist ein Folgepunkt weiter vom Ursprung entfernt als e_{max}, so gehen wir davon aus, daß für den Startwert die Folge gegen unendlich strebt, also bestimmt divergiert.
- Ist umgekehrt ein Folgepunkt weniger als e_{min} vom Ursprung der komplexen Zahlenebene entfernt, so konvergiert die Folge gegen Null.
- Kann nach einer maximalen Anzahl n_{max} von Durchgängen immer noch keine Aussage über Divergenz oder Konvergenz getroffen werden, so haben wir eine zyklische Folge vor uns.

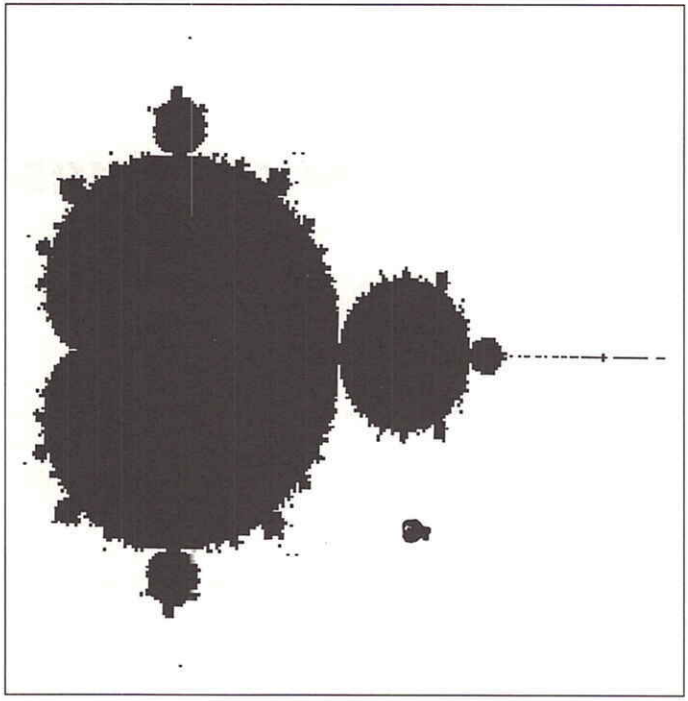
Das zweite Kriterium wird uns, wie wir noch sehen werden, nicht viel helfen, denn meist konvergiert die Mandelbrot-Folge nicht gegen (0,0), sondern entweder gegen Fixpunkte nahe am Ursprung oder sie bildet eine zyklische Folge. Folgendes Verhalten ist also sinnvoll zu unterscheiden:

- Divergenz (Die Folgeglieder entfernen sich vom Ursprung);
- zyklisches Verhalten (Die Folge springt immer wieder zu den gleichen Punkten. Die Anzahl der Punkte gibt die Ordnung des Zyklus an. Dieses Verhalten kann auch so gesehen werden, daß die Folge in diesem Fall einen ausgedehnten Attraktor, der aus mehreren Punkten besteht, besitzt).
- Konvergenz gegen einen Fixpunkt, für den gilt: $z_{n+1} = z_n$.

Die Formel von Mandelbrot hat also meist nur zwei Attraktoren, von denen der eine Unendlich ist.

4.1.1 Die Mandelbrot-Menge

Wie oben bereits angesprochen, haben wir vier Einstellungen vorzunehmen, die das Aussehen des Endbildes bestimmen. Wir haben sozusagen in der Formel z^2-c eine Funktion mit vier Parametern vor uns. Bei der Darstellung in einer Ebene können wir dabei nur zwei davon variabel wählen und müssen die anderen beiden festhalten. Wir legen Quasi-Schnitte in ein vierdimensionales Gebilde. Diese Betrachtung sollte aber nicht sofort verwirren, denn zur Anschauung trägt sie nicht besonders bei!
Bei seinen Untersuchungen hielt Mandelbrot den Startwert der Folge $z_\emptyset$ fest und variierte den komplexen Parameter c. Dabei wählte er $z_\emptyset$ und setzte c gleich den Koordinaten des zu untersuchenden Bildpunktes. Er prüfte dann das Verhalten der Folge gegen unendlich. Dabei erhielt er eine Menge, die heute nach ihm Mandelbrot-Menge genannt wird (siehe Abbildung 4-1-1). Als er sie zum erstenmal sah, traute er sicher seinen Augen nicht, denn das Ergebnis muß doch ein wenig überraschend gewesen sein!



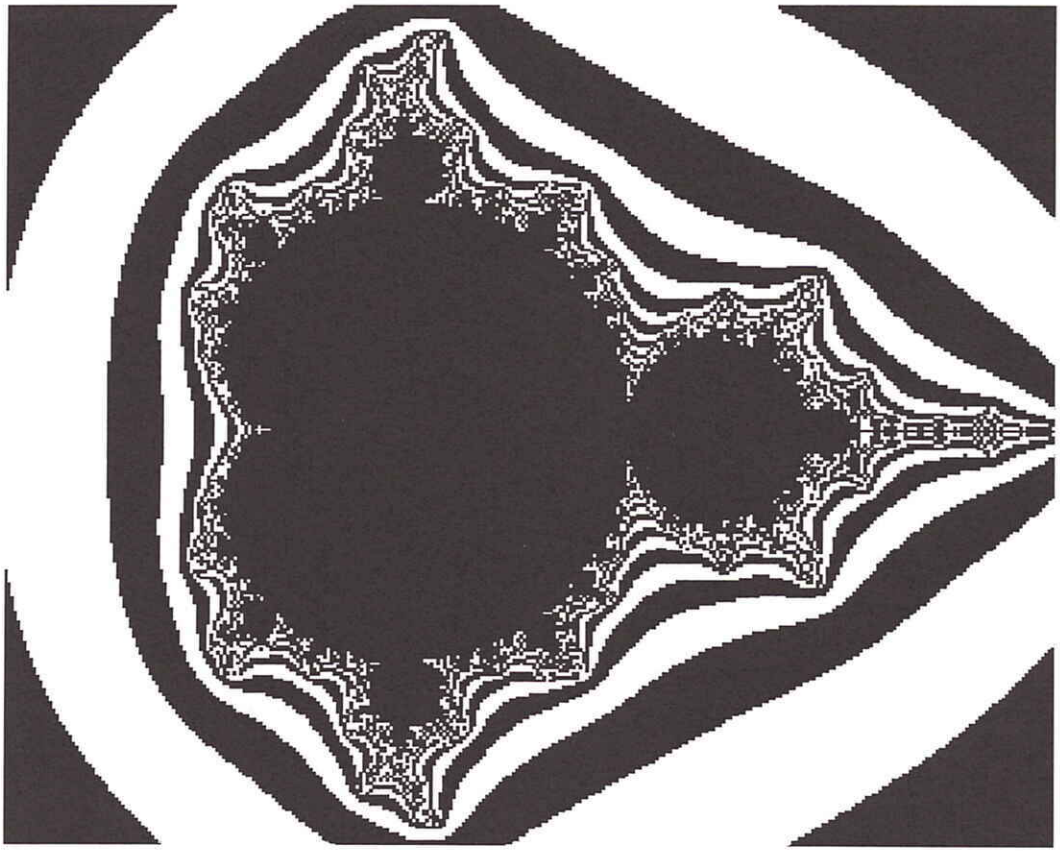
4-4-1 Die Mandelbrot-Menge

Die Mandelbrot-Menge M_{Mandel} besteht also aus den Werten von c, für die die Rückkopplung nicht divergiert:

M_{Mandel} = (c aus $\mathbb{C} | z_n$ divergiert nicht)
(C ist die Menge der komplexen Zahlen)

Sie beinhaltet also sowohl zyklische Folgen, als auch konvergentes Verhalten. Diese Menge ist heute das Markenzeichen der experimentellen Mathematik geworden und hat immer wieder etwas Faszinierendes an sich, gerade wenn man weiß, wie einfach der Entstehungsprozeß ist. Eine Bremer Forschungsgruppe hat ihr den zutreffenden Spitznamen Apfelmännchen verpaßt, unter dem sie auch bekannt geworden ist.

Ein Algorithmus für die Berechnung könnte also wie folgt aussehen:



4-1-2 Geschwindigkeits-Diagramm (Höhenlinien)

1) Wahl der komplexen Zahlenebene
2) Für jeden Punkt dieser Ebene werden folgende Anweisungen durchlaufen:

   - Startwert $z_{\emptyset} = \emptyset$ und c gleich den Punktkoordinaten setzen.
   - Solange die Folgenwerte z_{n+1} berechnen, bis:
     z_{n+1} größer als e_{max} ist, also die Folge divergiert,
     n_{max}-Durchläufe berechnet sind.
   - Falls n_{max}-Durchläufe bis zum Abbruch benötigt wurden, haben wir einen Punkt aus dem Inneren des Apfelmännchens vor uns. Wir färben diesen Punkt schwarz (Hintergrundfarbe), ansonsten weisen wir ihm eine Farbe gemäß der benötigten Durchläufe n zu. Am besten Farbe des Punktes gleich n `modulo` Anzahl der Farben.

Ein zusätzliches Abbruchkriterium nach e_{min} zeigt in der fertig berechneten Grafik im Inneren des Apfelmännchens an exponierten Stellen gesetzte Punkte (zum Beispiel im Zentrum der Zykloide und des Kreises – siehe Optimierung der Berechnung). Nur für diese Werte konvergiert die Folge also scheinbar gegen den Ursprung oder gegen Punkte nahe bei ihm. Deshalb kann dieses Abbruchkriterium getrost weggelassen werden. Die Wahl von n_{max} bestimmt die Schärfe der Begrenzung zwischen Mandelbrot-Menge und Divergenzbereich. Wollen wir Vergrößerungen

aus dem Randgebiet vornehmen, so müssen wir n_{max} möglichst groß wählen. Daß n_{max} nicht beliebig groß gewählt werden kann ist klar, denn wir wollen ja die Zeit zur Berechnung in endlichen Werten halten! Die Rechenzeiten für die Bilder liegen im Schnitt zwischen wenigen Minuten und einem Tag.

Zeichnen wir nur die Werte für die wir n_{max} Berechnungen benötigt haben, so erhalten wir die eigentliche Mandelbrot-Menge. Werten wir dagegen auch die Punkte aus für die die Folge divergiert, so erhalten wir sozusagen Geschwindigkeitsdiagramme (oder auch die »Höhenlinien des Apfelmännchens« (siehe Abbildung 4-1-2).

In ihnen kann man über den Verlauf der Farben darauf schließen, wie lange es gedauert hat, bis die Folge abgebrochen wurde.

Peitgen und Richter (Uni Bremen) haben folgende interessante Tatsache festgestellt: Die sich bei dieser Geschwindigkeitsauswertung ergebenden Linien sind die Äquipotentiallinien eines elektrischen Feldes, die sich bei einem geladenen Metallkörper einstellen würden, wenn dieser die Form eines Apfelmännchens hätte.

Sollten Sie auch andere Literatur zur Mandelbrot-Menge studiert haben ist Ihnen vielleicht aufgefallen, daß andere Autoren nicht die Formel z^2-c, sondern z^2+c verwenden. Daraus resultiert, daß das Apfelmännchen an der Imaginärachse gespiegelt erscheint. Wollen Sie solche Bilder generieren, wie Sie sie in diesen Büchern finden, so müssen Sie den Realteil der gegebenen Koordinaten mit -1 multiplizieren! Außerdem ist dort meist die Imaginärachse nach oben orientiert, während sie bei mir nach unten weist.

Interessante Stellen in der Mandelbrot-Ebene, die es wert sind vergrößert zu werden, liegen am Rand des Apfelmännchens. Dabei ist es sinnvoll, kleine Apfelmännchen, die vor einem großen liegen, zu vergrößern, Punkte, die noch zur Mandelbrot-Menge gehören, doch völlig isoliert erscheinen (»Täler«) und die Verbindung zwischen isolierten Punkten und dem Hauptkörper (sozusagen die tiefen »Schluchten«, die diese »Täler« verbinden) zu untersuchen.

Bei der Untersuchung der »Täler« und der weiter entfernten Punkte werden wir feststellen, daß der Körper des Apfelmännchens, also die Mandelbrot-Menge, zusammenhängend ist. Dabei sind die Verbindungslinien zu weit entfernten Punkten unendlich dünn und verkrumpelt, also fraktal. Wir werden auch bei der Untersuchung der Knospen und weit entfernten Ableger immer wieder Gebilde finden, die der Ausgangsfigur der Mandelbrot-Menge bis ins Detail gleichen. Wir stoßen hier wieder auf einen eindeutigen Beweis für Fraktale, die Selbstähnlichkeit. Diese ist so vollkommen, daß es kaum glaubhaft erscheint! Die Vergrößerungen, die wir bei unserer Entdeckungsreise vornehmen werden, gehen schnell ins Astronomische. Würden wir, nachdem wir mehrere Male in die komplexe Ebene gezoomt haben, das gesamte Apfelmännchen zeichnen, so würde dies leicht eine Fläche der Größe eines Fußballplatzes einnehmen.

Interessant ist auch der Vergleich zwischen dem Grand-Canyon in den USA und dreidimensionalen Vergrößerungen des Apfelmännchens. Betrachtet man ein Bild des tiefen Tals, das der Colorado gegraben hat, so ist die Ähnlichkeit erstaunlich. Hier finden wir die Ausläufer der Hochebene, die in den Rand hineinragen, um dann senkrecht mehrere hundert Meter abzubrechen. Wir erkennen die vielen Seitentäler, die sich zu kleinen

Canyons erweitern. Gerade Luftaufnahmen belegen die selbstähnlichen Strukturen des fraktalen Canyon-Rands. Die Verwandtschaft ist eigentlich nicht weiter erstaunlich, denn wie beim Apfelmännchen hängen aufgrund ihres Entstehungprozesses die Seitentäler mit dem Haupttal zusammen. Wir haben damit ein beeindruckendes Beispiel für den Zusammenhang zwischen fraktalen, mathematischen Mengen und der Natur vor uns. Oder klingt das zuweit hergeholt?

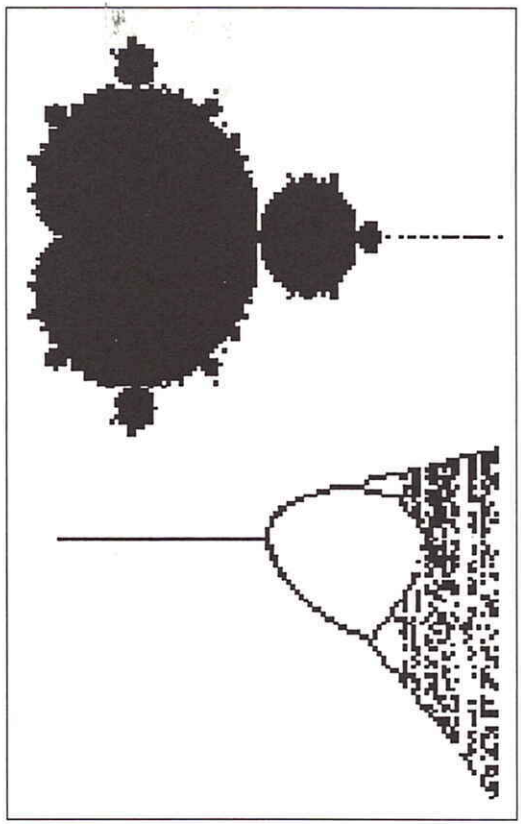
Noch einige Anregungen und Hinweise zum Experimentieren:

- Wählen Sie $z_\emptyset$ nicht wie bisher gleich Null, sondern setzen Sie z_n gleich einer komplexen Zahl. Dabei ist zu beachten, daß für $|z_\emptyset| > 1$ die Folge niemals konvergieren kann. Deshalb ist z_0 aus einem Kreis mit dem Radius 1 zu wählen.
- Werten Sie doch nach anderen Kriterien aus (bisher verwendeten wir den Abstand re^2+im^2 vom Ursprung als Maß für einen Abbruch):

- $|re| + |im| > e_{max}$?
- $|re| > e_{max}$?
- $|im| > e_{max}$?
- Maximum($|re|$ oder $|im|$) $> e_{max}$?

4.1.2 Zusammenhang zwischen Apfelmännchen und Feigenbaum-Diagramm

Bei den Feigenbaum-Diagrammen haben wir bereits das Verhalten einer reellwertigen quadratischen Rückkopplung untersucht. Dabei haben wir die sogenannten Feigenbäume erhalten. Sie zeichneten sich durch periodenverdoppelnde Kaskadenverzweigung aus. Die Formel für die Rückkopplung könnte zum Beispiel wie folgt lauten:



4-1-3 Zusammenhang zwischen Mandelbrot-Menge und Feigenbäumen

$$p_{n+1} = p_n^2 - k$$

Vergleichen wir diese Formel mit der Mandelbrot-Gleichung

$$z_{n+1} = z_n^2 - c,$$

so fällt die Ähnlichkeit sofort auf. Deshalb ist es auch nicht weiter erstaunlich, daß die Verzweigungsstellen des Feigenbaums mit den Knospen des Apfelmännchens auf der reellen Achse korrespondieren (siehe Abbildung 4-1-3).

Wie uns bereits bekannt ist, verzweigt der Feigenbaum im Chaosbereich unendlich oft. Nach jeder Verzweigung erhalten wir wieder ein kleines Feigenbäumchen, daß seinem großen Vorbild bis ins Detail gleicht. Damit

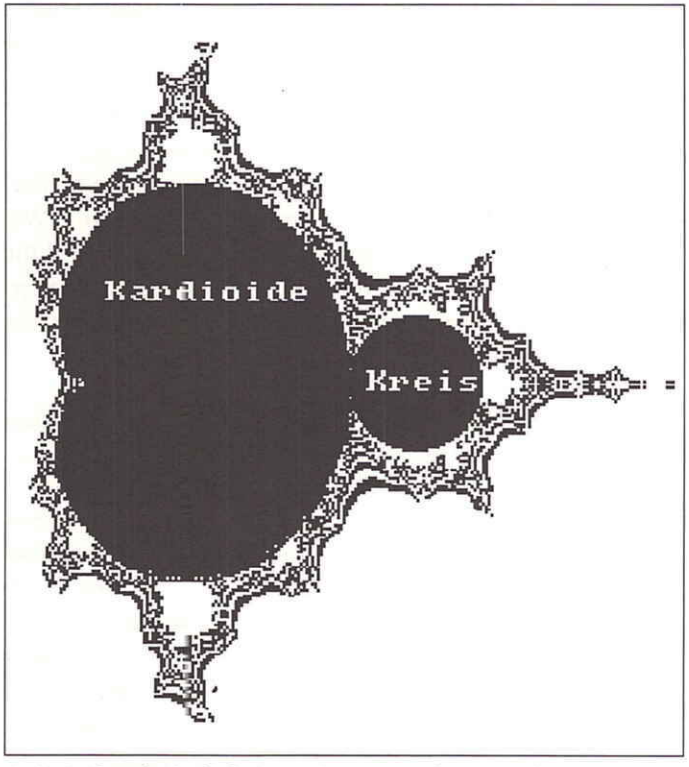
kann es uns jetzt auch nicht mehr verwundern, daß auf der Spitze des Apfelmännchens unendlich viele kleine »Apfelsöhnchen« aufgespießt sind, die der Ausgangsfigur gleichen.

4.1.3 Geschwindigkeitsoptimierung der Berechnung des Apfelmännchens

Die lange Rechenzeit bei der Auswertung der Folge z^2-c wird sehr schnell störend, gerade wenn man tiefer in die komplexe Zahlenebene eindringt, also sehr starke Vergrößerungen vornimmt. Gerade für den uninteressanten Teil des Bildes, das schwarze Innere des Apfelmännchens, braucht die Berechnung am längsten. Dies liegt natürlich daran, daß für die meisten der Punkte im Inneren des Apfelmännchens keine eigentliche Konvergenz gegen Null vorliegt, sondern eine periodische Folge auftritt. Deshalb wird die Berechnung für Startwerte aus dem eigentlichen Apfelmännchen-Körper erst nach n_{max}, der maximalen Anzahl der Berechnungen abgebrochen.

Dieser Nachteil kann durch Berücksichtigung der Form des Körpers erheblich behoben werden (siehe Abbildung 4-1-4). Wir erkennen im vorderen Teil ein kreisförmiges Gebilde mit dem Mittelpunkt $(1,0)$ und dem Radius $\frac{1}{4}$.

Diese Eigenschaft kann zur Geschwindigkeitsoptimierung genutzt werden. Wir testen einfach bevor wir die Berechnung der Folge starten, ob der Punkt innerhalb dieses Kreises liegt. Wenn ja, dann können wir sicher sein, daß der Endwert ebenfalls in ihm liegen wird und wir brauchen daher die Berechnung nicht mehr auszuführen. Dieses Vorgehen bringt eine gewaltige Geschwindigkeitssteigerung für Bilder, die große Teile des Hauptkörpers beinhalten.



4-1-4 Geschwindigkeitsoptimierung der Berechnung

Auch für den großen Teil des Apfelmännchen-Inneren kann eine geometrische Figur gefunden werden, die diesen ungefähr beschreibt. Er kann als eine Zykloide (Rollkurve) beschrieben werden. Der Test verläuft genauso wie bei dem Kreis und trägt damit auch zur Geschwindigkeitsoptimierung bei. Eine Zykloide entsteht beim Abrollen eines Kreises auf einem anderen, wobei die Bewegung eines fest gewählten Punktes auf dem äußeren betrachtet wird. In unserem Fall rollen zwei Kreise mit dem Radius $\frac{1}{4}$ aufeinander ab und wir betrachten einen Punkt auf dem Rand des äußeren. In Parameterdarstellung lautet die Gleichung der Zykloide:

$$z(æ) = \frac{1}{2} * \exp(i * æ) - \frac{1}{4} * \exp(2 * i * æ)$$

Dabei durchläuft æ Werte von 0 bis $2*\pi$.

Unsere Zykloide ist ein Sonderfall. Sie ist eine Hypozykloide, da der zweite Kreis nicht in (Zykloide), sondern auf dem ersten abrollt. Außerdem bezeichnet man sie, da beide Kreise gleich groß sind, auch als Kardioide (Herzkurve). Wir können ihre Bestimmungsgleichung auch nach Real- und Imaginärteil wie folgt schreiben:

$$\mathrm{Re}(z(æ)) = \frac{1}{4} * (2*\cos(æ) - \cos(2*æ))$$

$$\mathrm{Im}(z(æ)) = \frac{1}{4} * (2*\sin(æ) - \sin(2*æ))$$

oder auch

$$\left(\mathrm{Re}^2 + \mathrm{Im}^2 - \frac{1}{16}\right)^2 = \frac{1}{4} * \left(\left(\mathrm{Re} - \frac{1}{4}\right)^2 + \mathrm{Im}^2\right)$$

Zum Testen, ob ein Punkt z=x+y*i innerhalb des Kreises oder der Kardioide liegt, geht man folgendermaßen vor:

für den Kreis

$$(x-1)^2 + y^2 \leqq \frac{1}{16}$$

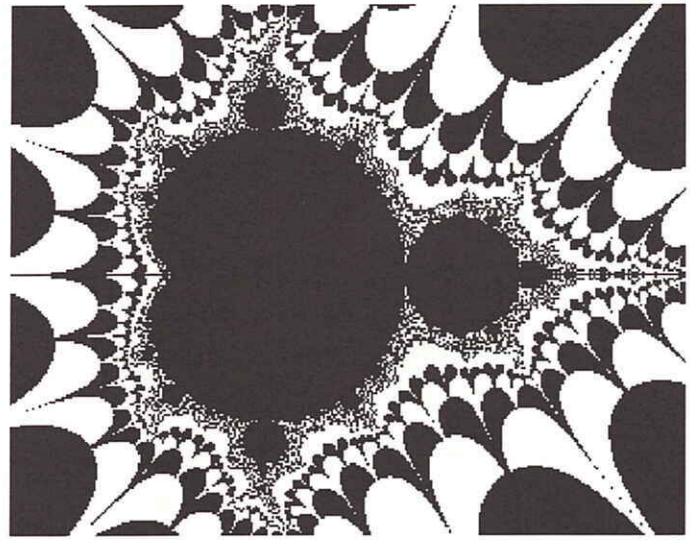
für die Kardioide

$$r=x^2+y^2 \text{ und } s = \sqrt{\left(r + \frac{x}{2} + \frac{1}{16}\right)}$$

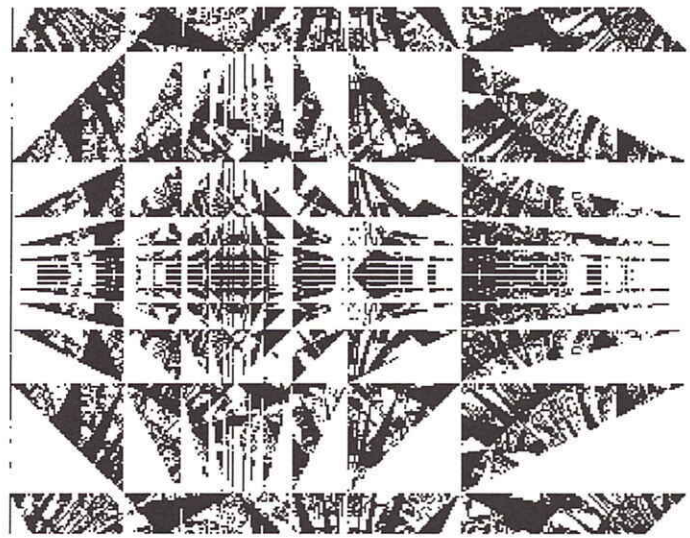
$$16*r*s \leqq 5*s+4*x+1$$

Bevor wir den Algorithmus starten, testen wir für jeden Punkt vor jedem Durchlauf, ob seine Koordinaten eine dieser beiden Ungleichungen erfüllt. Wenn ja, dann wissen wir, daß n_{max}-Durchläufe benötigt werden, also erhält der Punkt die Farbe Schwarz.

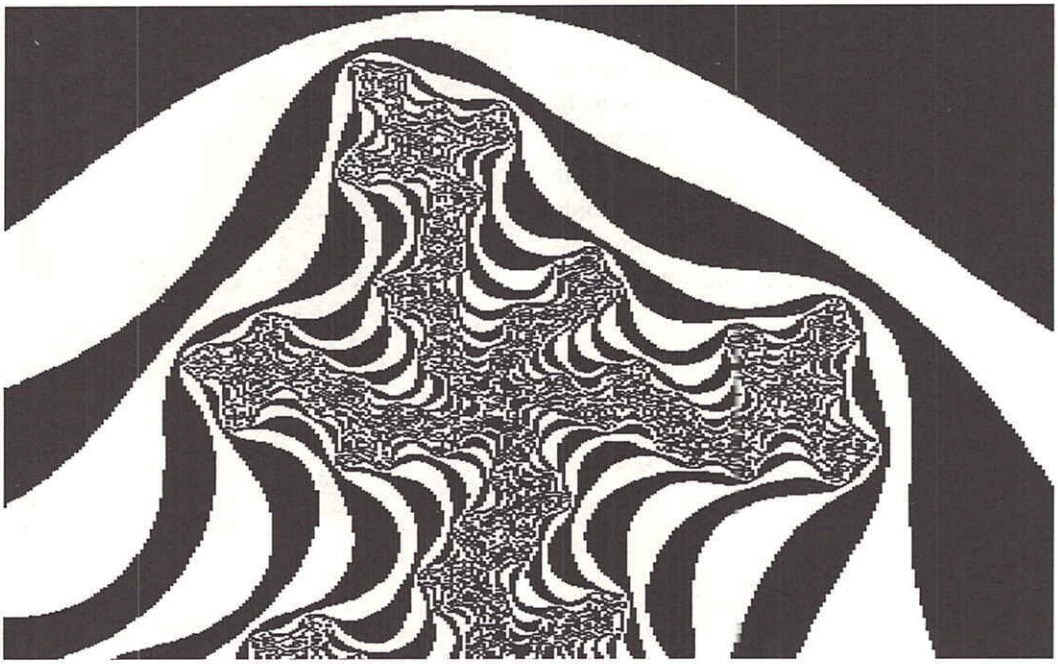
Der Kreis und die Zykloide beschreiben nicht die Randbereiche des Apfelmännchens! Deshalb bringt diese Art der Optimierung nur Vorteile, solange große Teile des Hauptkörpers im Bild erscheinen. Außer der Einbeschreibung des Kreises und der Zykloide kann die Berechnung auch durch die Berück-



4-1-5 (1) $|y|>e_{max}$



4-1-5 (2) $|x|+|y|>e_{max}$



4-1-6 (1)

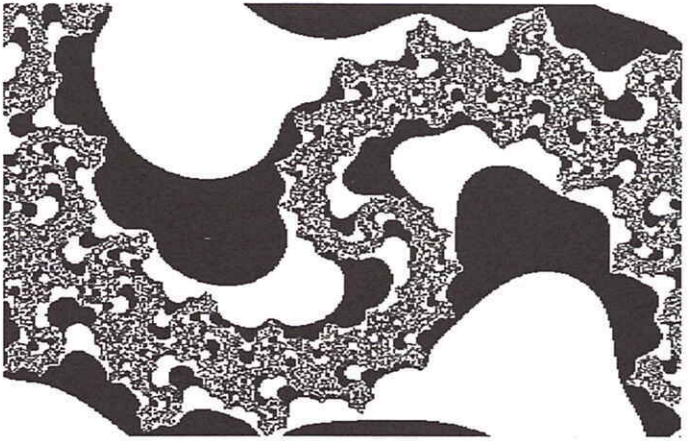
sichtigung der Achsensymmetrie zur reellen Achse weiter beschleunigt werden.
Interessant ist das Verhalten der Mandelbrot-Rückkopplung für Punkte aus dem Inneren des Apfelmännchens. Wird z aus dem Kreis gewählt, so erhalten wir einen attraktiven Zyklus der Ordnung 2, das heißt, zwei Punkte werden immer nacheinander angenommen. Liegt die komplexe Zahl z in dem Zykloid, so bewegen sich die Folge-Punkte auf einen Fixpunkt zu. Ein weiteres interessantes Verhalten ergibt sich für Punkte aus dem verkleinerten »aufgespießten« Apfelmännchen (z = 1.75) oder aus den beiden großen Auswüchsen »Knospen« am Hauptkörper. Für sie ergibt sich eine zyklische Folge der Ordnung 3. Außerdem lassen sich auch Gebiete mit Zyklen der Ordnung 4, 5, 6 usw. finden.

Mehr ist nicht nötig! Lassen Sie sich von der Faszination der Bilder gefangennehmen und urteilen Sie selbst!
Die Abbildungen 4-1-6 (1) bis (7) und 4-1-7 zeigen Ausschnittsvergrößerungen des Apfelmännchenrandes.

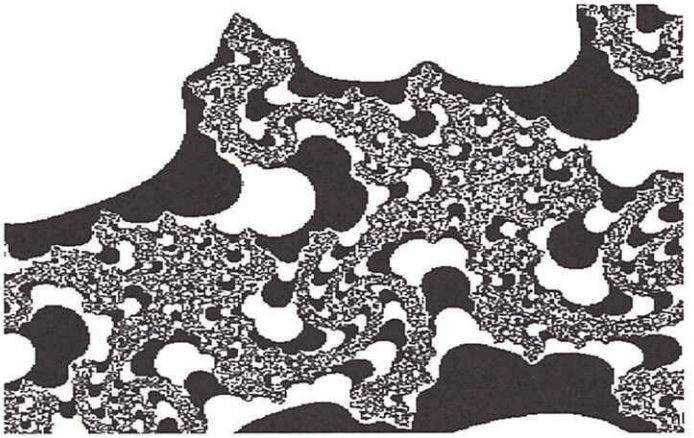
4.1.4 Julia-Mengen

Bis jetzt haben wir von den sechs möglichen Variationen der vier Anfangswerte erst eine genutzt. Werden die Rollen von c und z_0 vertauscht, so erhalten wir sogenannte Julia-Mengen. Sie haben ihren Namen von dem französischen Mathematiker Gaston Julia, der bereits 1920 mit Pierre Fatou auf dem Gebiet der Iteration komplexer Funktionen beachtliche Vorarbeit geleistet hat. Allgemein bezeichnet man ihm zu Ehren alle dynami-

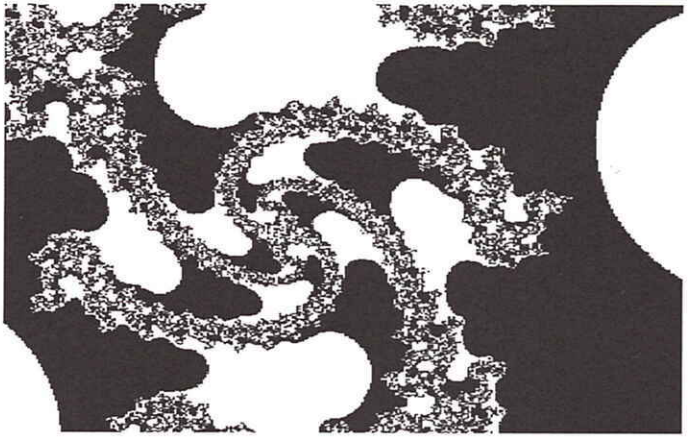
4-1-6 (2)



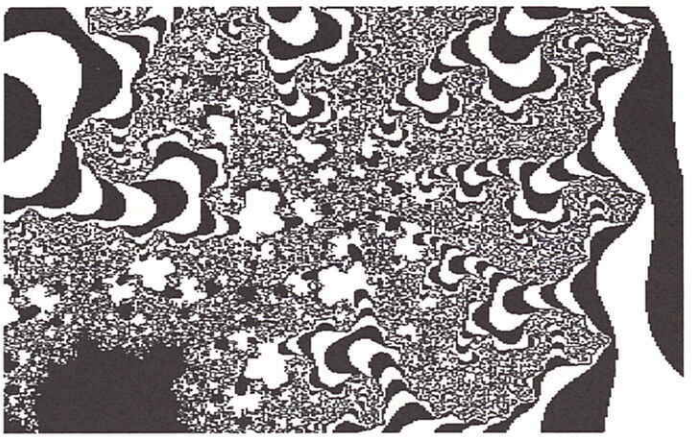
4-1-6 (3)



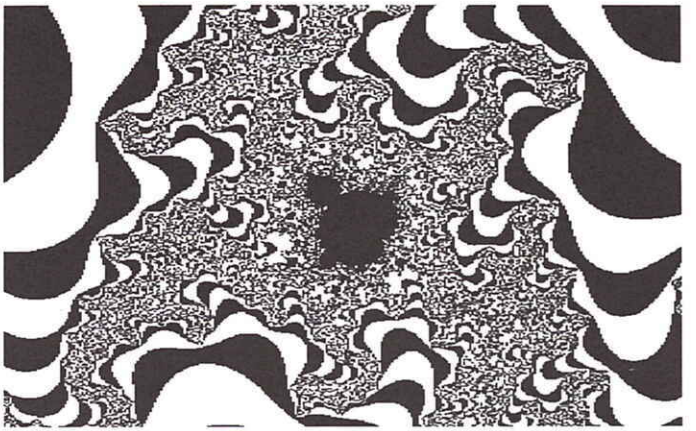
4-1-6 (4)



4-1-6 (5)



4-1-6 (6)



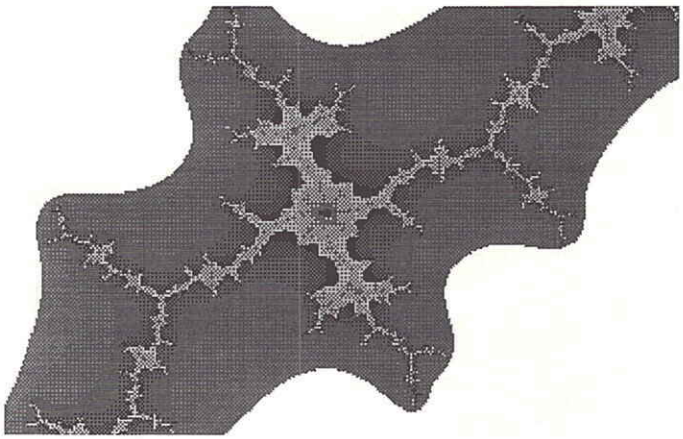
4-1-6 (7)

4-1-7 (1) Täler des Grand Canyon?

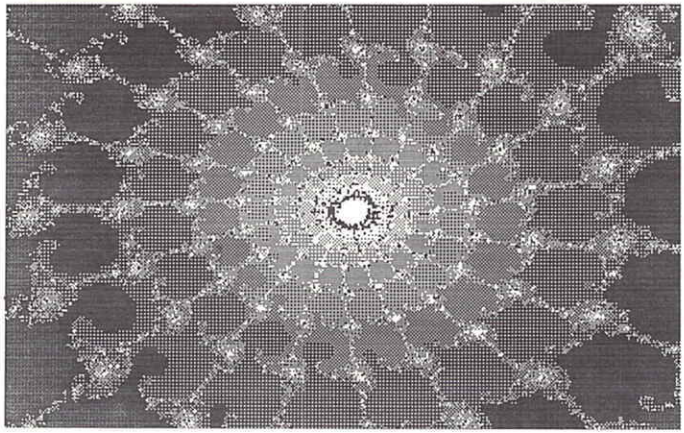
4-1-7 (3) Vergleich zur Grau-Interpretation

4-1-7 (4) Ein »Apfelmännchen« auf der Spitze der Mandelbrot-Menge

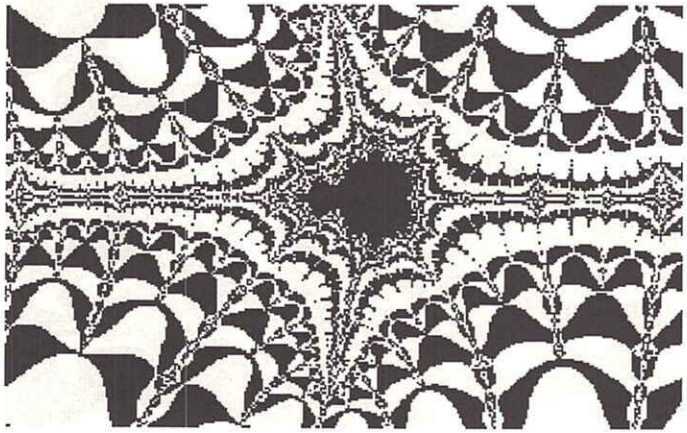
4-1-7 (5) Auf der Spitze der Mandelbrot-Menge



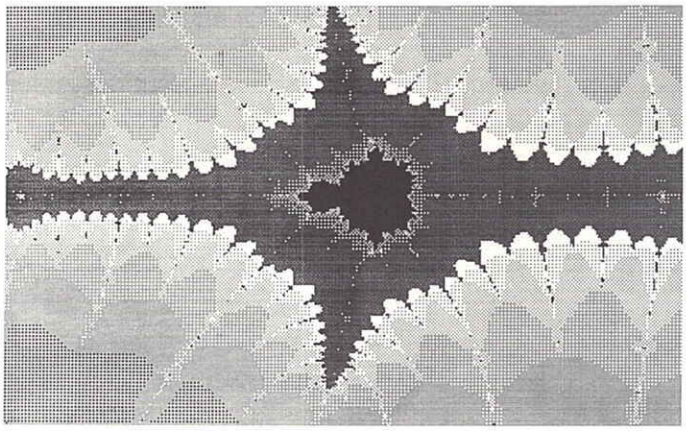
4-1-7 (1)



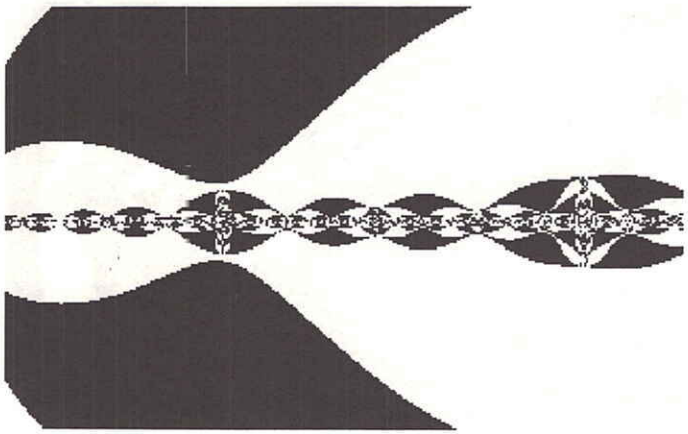
4-1-7 (2)



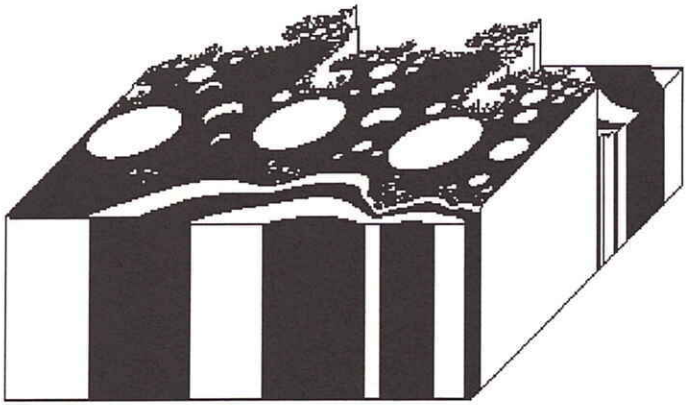
4-1-7 (3)



4-1-7 (4)



4-1-7 (5)



4-1-8 3-D-Interpretation einer Vergrößerung der Mandelbrot-Menge

schen Systeme die fraktale Grenzen besitzen, als Julia-Mengen.

Haben wir bei der Mandelbrot-Menge in der c_{im}-c_{re}-Ebene gezeichnet, so zeichnen wir nun in der Re-Im-Ebene. Wir wählen zu Beginn einen bildbestimmenden Wert für c fest, den wir während der Berechnung nicht verändern. »$z_{\emptyset}$« wählen wir dann in Abhängigkeit des zu berechnenden Bildpunktes.

Arbeiten wir zum Beispiel mit einer Auflösung von 320x200 Punkten und haben einen Wertebereich der reellen Achse von r_{min} bis r_{max} und einen der imaginären Achse von i_{min} bis i_{max}, so erhalten wir die Koordinaten und damit $z_{\emptyset}$ wie folgt:

$$st_x = \frac{(r_{max}-r_{min})}{320}$$

Werteabstand pro Pixel in x-Richtung

$$st_y = \frac{(i_{max}-i_{min})}{200}$$

Werteabstand pro Pixel in y-Richtung

Die Koordinaten des Punktes (50,100) ergeben sich zu:

$P(r_{min}+50*st_x, i_{min}+100*st_y)$

oder

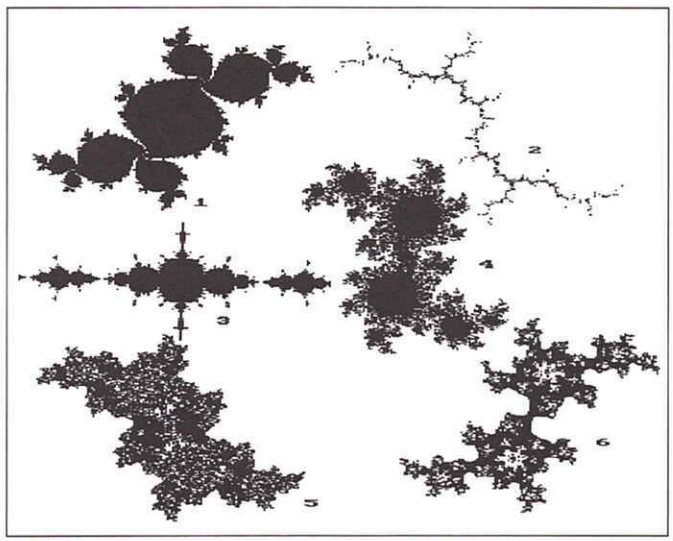
$z_{\emptyset} = (r_{min}+50*st_x) + (i_{min}+100*st_y)*i.$

Es ist klar, daß wir aufgrund der beliebigen (aber festen) Wahl von c unendlich viele verschiedene Julia-Mengen erhalten (einige Beispiele für Julia-Mengen zeigen die Abbildungen 4-1-9 und 4-1-10).

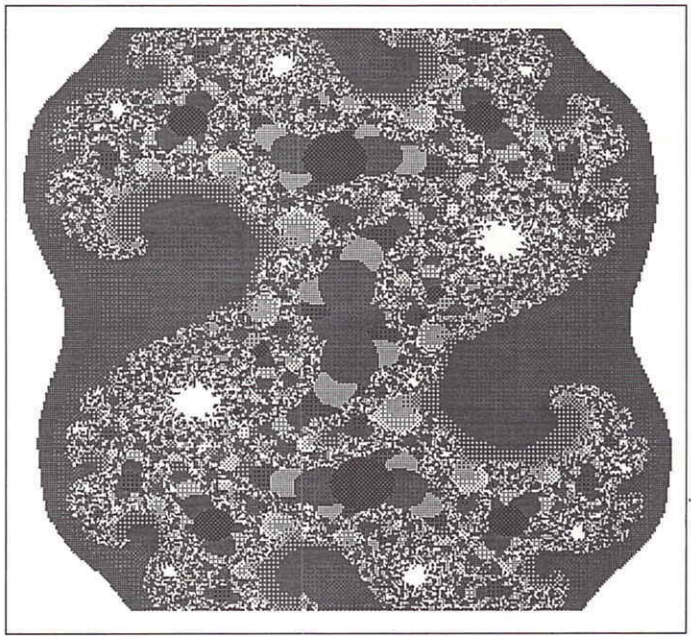
Dagegen gibt es für die feste Wahl von $z_{\emptyset}$ nur eine Menge, das heißt also, es gibt nur eine Mandelbrot-Menge. Die Untersuchung, ob ein Punkt zur Julia-Menge gehört oder nicht, wird analog wie bei der Mandelbrot-Menge überprüft. Ein Punkt gehört zur Julia-Menge J_c, wenn die Rückkopplung nach n_{max}-Schritten nicht gegen Unendlich gelaufen ist:

J_c = ($z_{\emptyset}$ geht nicht gegen Unendlich)

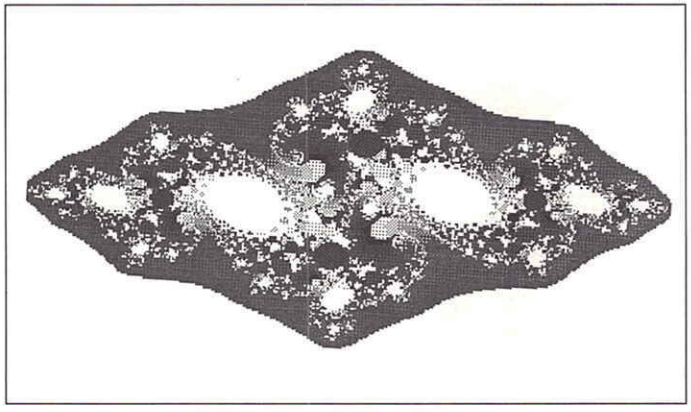
Eigentlich ist im mathematischen Sinne die Untersuchung erst nach unendlich vielen Schritten abzubrechen (also n_{max} unendlich zu setzen), aber die endliche Rechenzeit



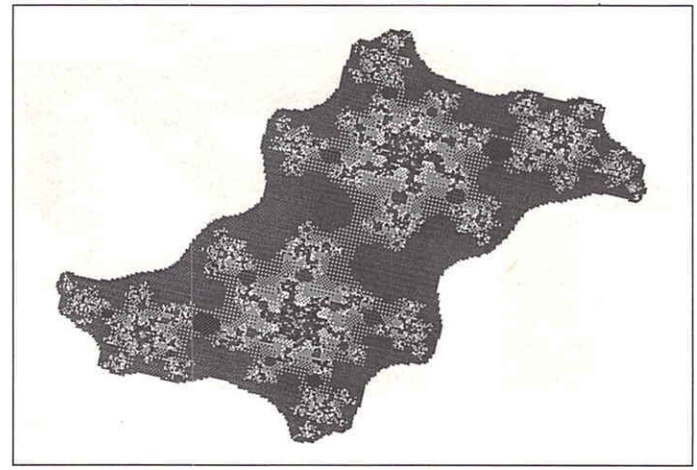
4-1-9



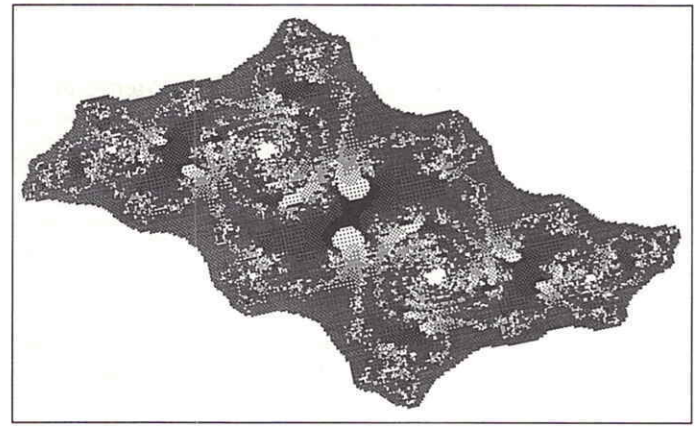
4-1-10 (1) Julia-Mengen



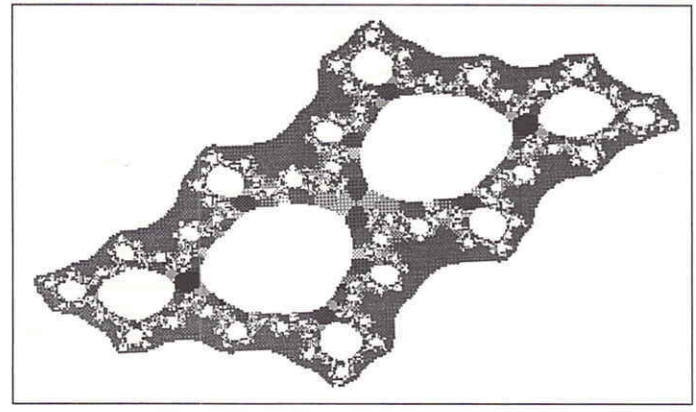
4-1-10 (2)



4-1-10 (3)



4-1-10 (4)



4-1-10 (5)

erfordert hier wieder die Beschränkung auf n_{max}-Schritte.

Ein Algorithmus für die Berechnung von Julia-Mengen könnte wie folgt aussehen:

1) Wahl der komplexen Zahlenebene und c.
2) Für jeden Punkt dieser Ebene folgende Anweisungen durchlaufen:
   - Startwert $z_{\emptyset}$ entsprechend den Punktkoordinaten setzen.
   - Solange die Folgenwerte z_{n+1} berechnen, bis z_{n+1} größer als e_{max} ist, also die Folge divergiert oder n_{max} Durchläufe berechnet sind.

Falls n_{max} Durchläufe bis zum Abbruch benötigt wurden, haben wir einen Punkt aus dem Inneren der Julia-Menge vor uns. Wir färben diesen Punkt schwarz (Hintergrundfarbe). Ansonsten weisen wir ihm eine Farbe gemäß der benötigten Durchläufe n zu. Am besten eignet sich: Farbe des Punktes gleich n `modulo` Anzahl der Farben.

Aussehen der Julia-Menge bei c=0
(siehe Abbildung 4-1-11 (1))

Für die Wahl $c=\emptyset$, also eine Rückkopplung der Form $z_{n+1} = z_n^2$ können wir uns das Aussehen der Julia-Menge recht einfach veranschaulichen:

- Für $|z_{\emptyset}|>1$ wird die Folge gegen Unendlich streben, also divergieren.
- Für $|z_{\emptyset}|<1$ wird die Folge gegen Null konvergieren.
- Für $|z_{\emptyset}|=1$ ergibt sich dann auch $|z_n|=1$. Die Julia-Menge ist also ein Kreis mit dem Radius 1.

Wählen wir c weiter vom Ursprung entfernt, so werden die Julia-Mengen immer interessanter. Wählen wir c aus dem Randbereich des Apfelmännchens, so erhalten wir die besten Bilder!

4.1.5 Zusammenhang zwischen Julia-Mengen und Apfelmännchen

Daß sowohl die Mandelbrot-Menge als auch die Julia-Mengen durch Auswertung der gleichen nichtlinearen Rückkopplung hervorgehen, ist nicht zu leugnen.

Es gibt tatsächlich einen recht erstaunlichen Zusammenhang zwischen diesen beiden Erscheinungsformen. Julia-Mengen sind für Werte von c nahe bei Null recht unergiebig. Erst wenn wir c aus dem Randbereich der Mandelbrot-Menge wählen, werden die Mengen immer interessanter. Betrachtet man eine Bildfolge in der c von Null bis in den Randbereich gewählt wurde, so erkennt man, wie sich die Darstellungsformen verändern. Es sieht so aus, als würde der Kreis an zwei Stellen immer weiter nach außen gezogen. Dabei löst sich der Rand der Menge immer mehr auf. Und nun das wichtigste: Wird c außerhalb der Mandelbrot-Menge gewählt, so hängt die berechnete Julia-Menge für diesen c-Wert nicht mehr zusammen. Je weiter entfernt wir c von dem Apfelmännchen-Körper wählen, desto zerstückelter ist J_c.

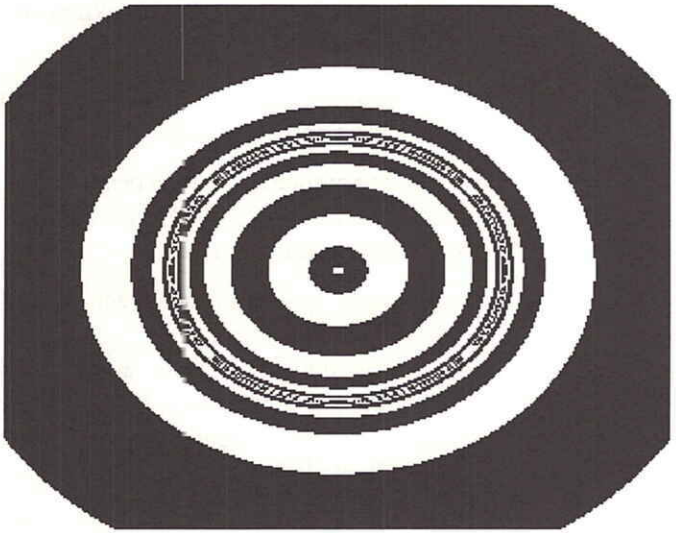
Aus diesen Erkenntnissen können wir folgende wichtige Schlußfolgerung ziehen:

Wird c aus der Mandelbrot-Menge gewählt, so hängt die dazugehörige Julia-Menge zusammen. Wählen wir dagegen ein außerhalb liegendes c, so zerfällt die Julia-Menge in mehrere Teile (siehe Abb. 4-1-11 (1-5).

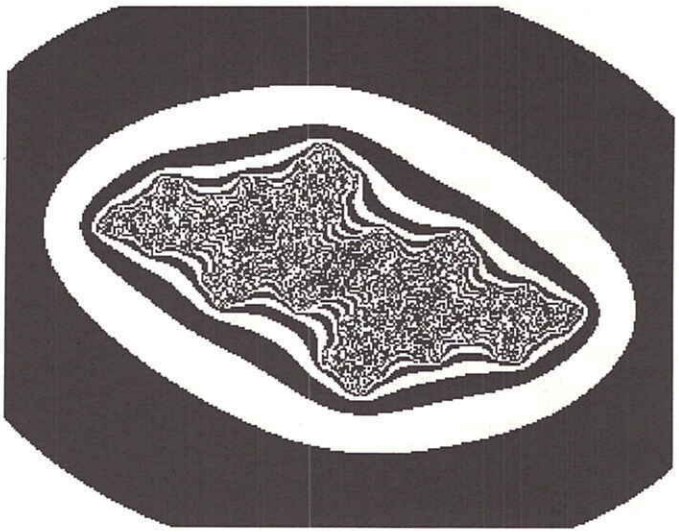
Diese Tatsache kann auch zur Definition des Apfelmännchens M herangezogen werden:

`M = (c aus C | J_c ist zusammenhängend)`

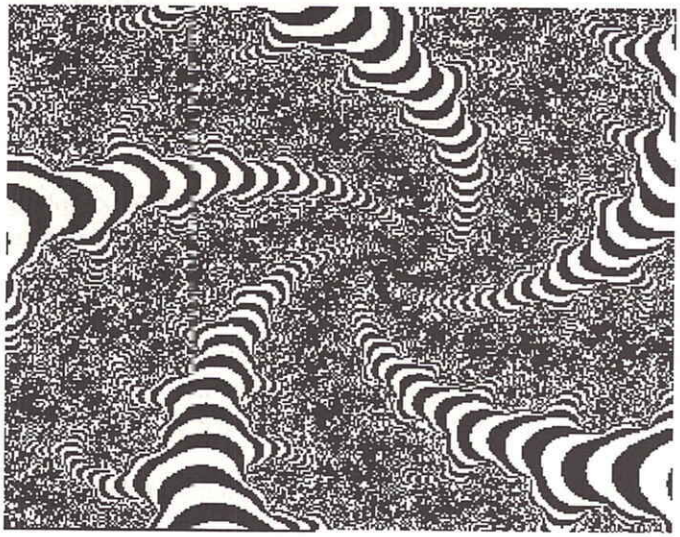
Dieser von uns empirisch gefundene Zusammenhang wurde in einem komplizierten mathematischen Beweis geklärt und ist damit auch von dieser Seite her abgesichert.



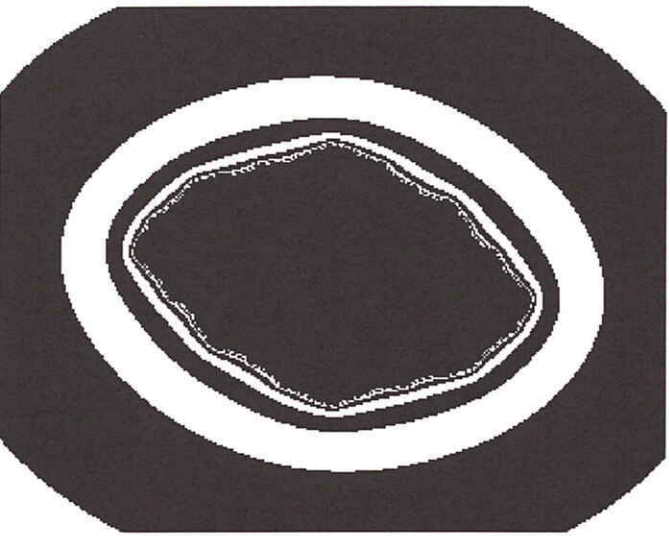
4-1-11 (1)



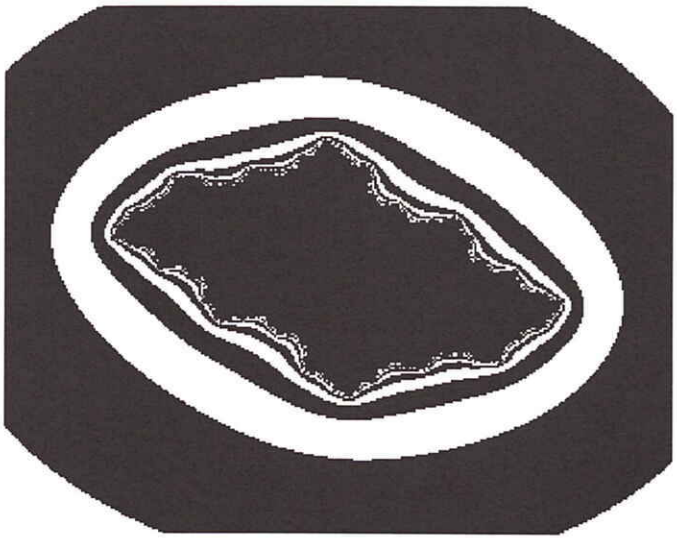
4-1-11 (4) (1)



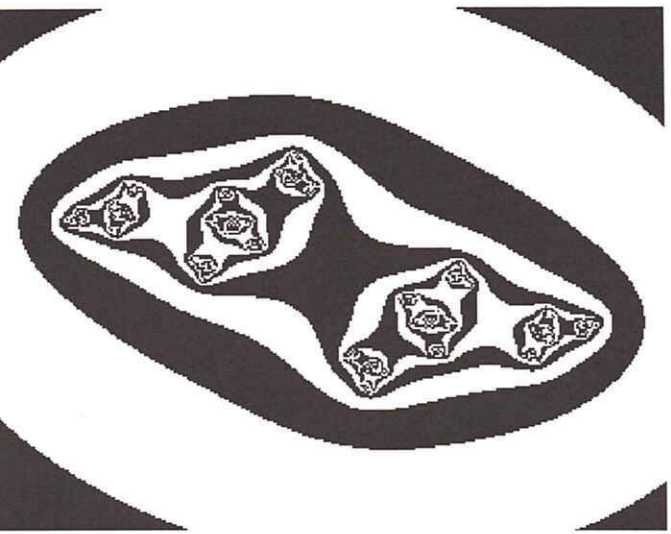
4-1-11 (4) (2)



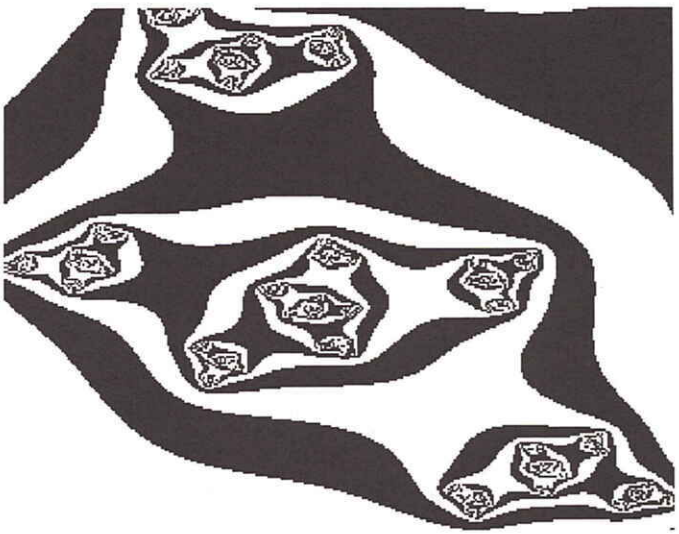
4-1-11 (2)



4-1-11 (3)



4-1-11 (5) (1)



4-1-11 (5) (2)

4.1.6 Symmetrie der Julia-Mengen

Es fällt bei der Betrachtung sofort auf, daß die Julia-Mengen bei einem Rotationswinkel von 180° drehsymmetrisch, also bezüglich des Ursprungs punktsymmetrisch sind. Dies liegt an der rein quadratischen Rückkopplung! (Quadrieren in der komplexen Ebene bedeutet eine Drehsymmetrie von 180°.)
Dies ist leicht zu verifizieren: Wie wir wissen, können wir unsere Rückkopplungsformel

$z_{n+1} = z_n^2 - c$ auch

$re_{n+1} = re_n^2 - im_n^2 - c_{re}$

und

$im_{n+1} = 2*re_n*im_n - c_{im}$

schreiben.

Setzen wir statt dem Punkt

(re_n, im_n) $(-re_n, -im_n)$ ein, so erhalten wir:

$re_{n+1} = re_n^2 - im_n^2 - c_{re}$

und

$im_{n+1} = 2*re_n*im_n - c_{im}$

Dies sind die gleichen Formeln wie für den Punkt (re_n, im_n). Wir erhalten also für beide Punkte die gleichen Folgewerte und damit auch die gleiche Farbe. Damit ist die Punktsymmetrie der Julia-Mengen begründet. Wählen wir c rein reell, so geht die Punktsymmetrie sogar in Achsensymmetrie bezüglich der Imaginärachse über. Berechnen wir eine Julia-Menge mit der Konstante $c=c_{re}+i*c_{im}$ und eine andere mit $c=c_{re}-i*c_{im}$, kehren also das Vorzeichen des Imaginärteils von c um, so geht die zweite Menge aus der ersten durch Spiegelung an der Real-Achse hervor.

Eine Geschwindigkeitsoptimierung bei der Berechnung einer Julia-Menge könnte also darauf aufbauen, nur eine Halbebene der komplexen Zahlenebene zu durchlaufen und dann aufgrund der Punktsymmetrie die andere durch Spiegelung der Berechneten zu zeichnen.

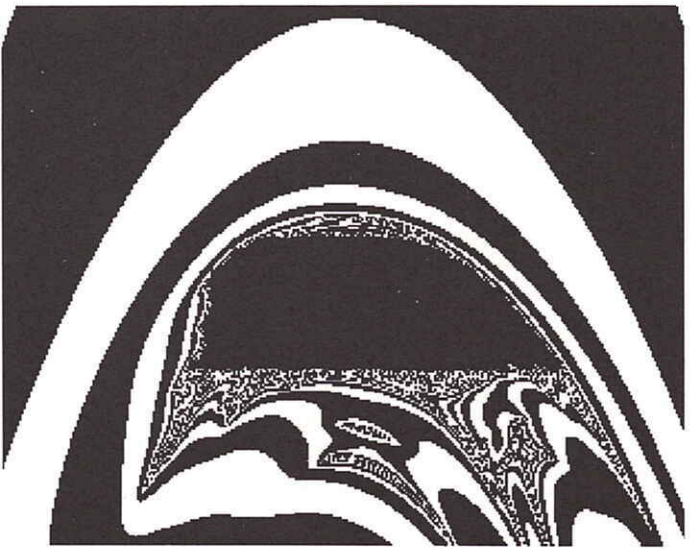
Zum Schluß noch ein Hinweis. Experimentieren Sie mit Vergrößerungen aus dem Randbereich der Julia-Mengen. Auch hier werden Sie die Selbstähnlichkeit entdecken. Egal wie tief Sie eindringen (allein begrenzt durch die Rechengenauigkeit des Computers!), Sie werden immer wieder gleiche Strukturen und Variationen desselben Themas finden. Der Rand der Julia-Menge ist also auch ein Fraktal. Diesem ordnen die Mathematiker eine fraktale Dimension zu, die gebrochen rational ist und zwischen eins und zwei liegt.

Warum es sinnvoll ist, eine solche Dimension einzuführen, werden wir einige Abschnitte später bei den geometrischen Fraktalen kennenlernen. Bisher ist noch kein allgemeines Berechnungsverfahren für die fraktale Dimension für Julia-Mengen bekannt. Ruelle (1981) hat aber für kleine Werte von c genaue Aussagen treffen können. Für Werte von c nahe dem Ursprung erhalten wir die Dimension $D(J_c)$

$$D(J_c) = 1 + \frac{|c|^2}{4*\log 2} + \text{Terme höherer Ordnung}$$

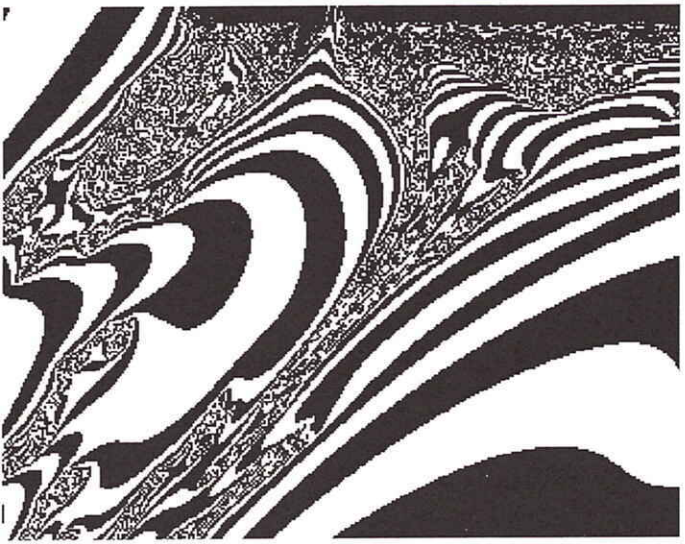
Für $c=0$ erhalten wir also die Dimension $D(J_c) = 1$. Dieses Ergebnis erstaunt uns nicht weiter, denn wir haben ja bereits weiter oben festgestellt, daß diese Julia-Menge ein einfacher Kreis mit dem Radius Eins ist.

Bei der Mandelbrot-Menge wurde schon angesprochen, daß die Auswertung Re^2+Im^2 nicht unbedingt benutzt werden muß. Auch bei den Julia-Mengen erhalten wir interessante Resultate, wenn wir mit dem Abbruchkriterium $Im^2>e_{max}$ arbeiten.



4-1-12 (1) »Schnitt« durch z^2-c

4-1-12 (2)



4-1-12 (3)

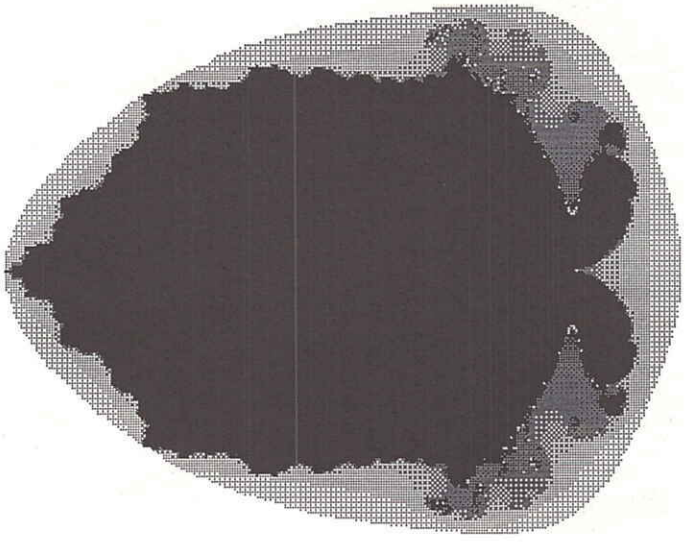
4.1.7 Weitere »Schnitte«

Von den sechs angesprochenen Möglichkeiten die Anfangsparameter zu variieren, haben wir nun bereits zwei kennengelernt und analysiert. Wie bereits weiter oben angesprochen, kann man sich die Feigenbaum-Formel als Funktion mit vier Parametern vorstellen. Von diesen vier Parametern können wir auf unserem Bildschirm immer nur zwei darstellen. Deshalb müssen immer zwei Parameter konstant gehalten werden. Wenn man so will, ziehen wir dadurch »Schnitte« durch eine vierdimensionale Funktion.

Mögliche Variationen der Anfangsparameter

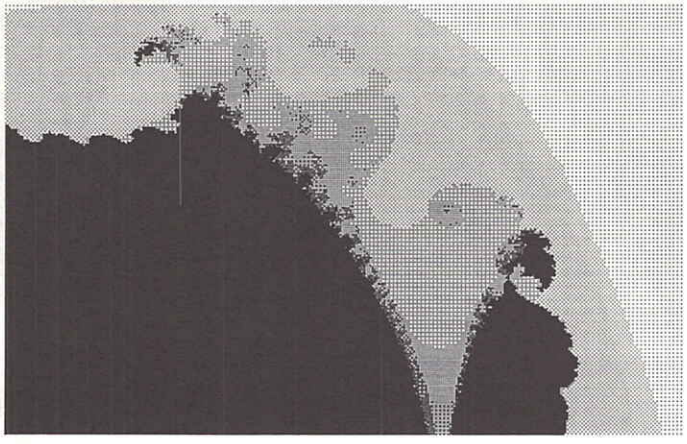
(`f` bedeutet: Wert für die Berechnung fest gewählt. `b` bedeutet: Wert wird durch die Koordinaten des Bildpunktes bestimmt.)

cre	cim	re	im	
b	b	f	f	(Apfelmännchen)
f	f	b	b	(Julia-Mengen)
f	b	f	b	(andere Schnitte)
b	f	b	f	
b	f	b	b	
f	b	b	f	



4-1-13 (1)

Eine neue fraktale Menge und Vergrößerungen aus ihrem Randbereich



4-1-13 (2)

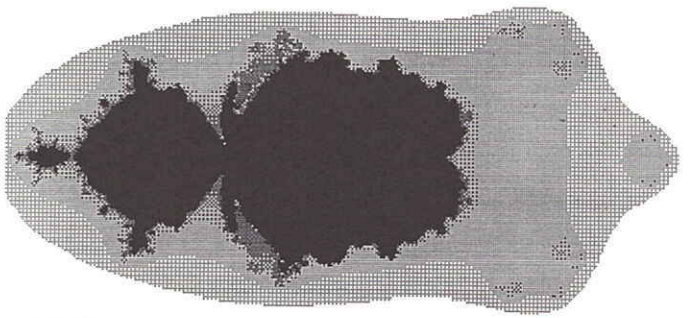
4-1-13 (3)

4-1-13 (4)

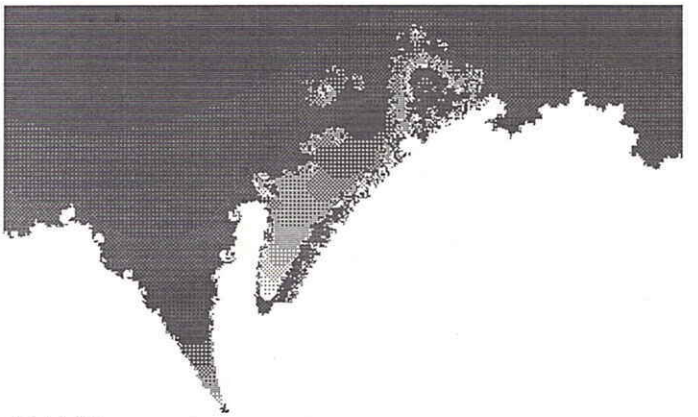
Die bisher von uns vorgenommenen Schnitte verliefen entweder parallel zur c-Ebene (Apfelmännchen) oder zur z-Ebene (Julia-Mengen). Die Bilder die wir dabei erhielten, zeichnen sich durch einen hohen Grad an Ordnung und Symmetrie aus. Nehmen wir nun andere Schnitte vor, so erhalten wir die wildesten Figuren (siehe Abbildungen 4-1-12 (1) bis (3) auf Seite 61). Sie sind Verzerrungen von Elementen der Julia-Mengen und des Apfelmännchens, denen die Symmetrie und Ordnung abhanden gekommen ist.

Bis jetzt sind vor allem die Julia-Mengen und die Mandelbrot-Menge untersucht worden. Gehen Sie doch einmal auf Entdeckungs-

Neue rekursive Folgen bedingen völlig unterschiedliche Mengen



4-1-14 (1)



4-1-14 (2)

reise, versuchen Sie die anderen Schnitte zu katalogisieren und neue Zusammenhänge zu finden. Es gibt hier noch viel Arbeit und offene Fragen! Wie steht es zum Beispiel mit der Selbstähnlichkeit bei Vergrößerung der Schnitte?

4.1.8 Mengen höherer Ordnung (z^3-c, z^4-c und z^5-c)

Wie bei der rekursiven Folge z^2-c haben wir auch bei Gleichungen höherer Ordnung vier Parameter zur Verfügung, die das Aussehen des Bildes bestimmen. Sie ergeben sich wieder aus Real- und Imaginärteil der komplexen Werte $z_\emptyset$ und c. Wir wollen weiterhin mit den beiden Abbruchkriterium wie bisher arbeiten:

- Entfernt sich der Punkt mehr als e_{max} vom Ursprung, so brechen wir die Berechnung ab und die Folge divergiert für diesen Anfangspunkt.
- Benötigen wir mehr als n_{max}-Durchgänge, so zeigt die Folge periodisches Verhalten. Dieser Anfangspunkt gehört zur gesuchten Menge!

Analog zur Wahl der Ausgangsparameter müssen wir also sechs Fälle unterscheiden:

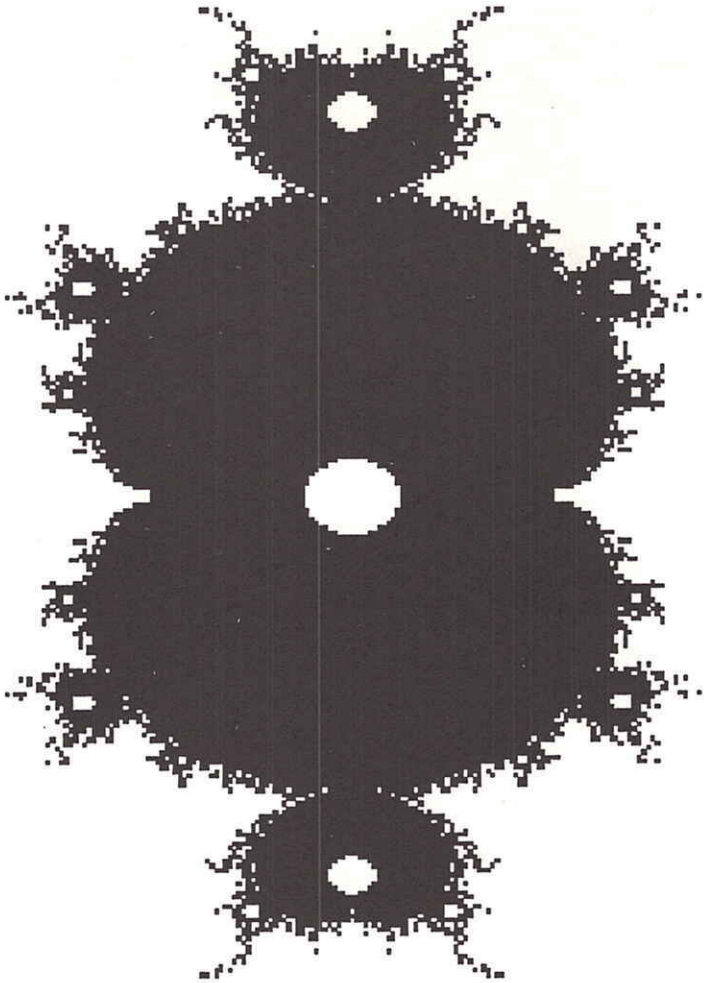
1) Auswertung analog wie bei der Mandelbrot-Menge, daß heißt $x_\emptyset$ wird fest gewählt und c durch den Bildpunkt bestimmt.

(Siehe zum Aussehen der Mengen höherer Ordnung die Abbildungen 4-1-15 (z^3-c), 4-1-16 (z^4-c) und 4-1-17 (z^5-c)).

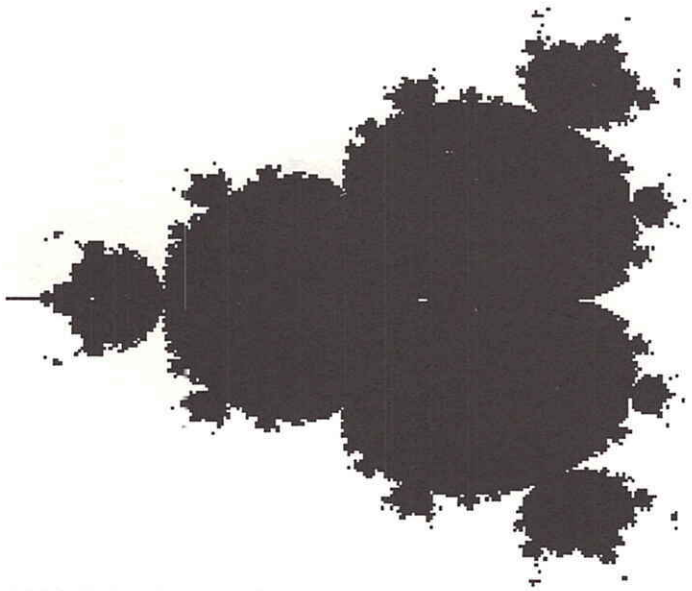
Wie bei der Mandelbrot-Menge werden wir bei dieser Art der Auswertung für die Funktionen z^3-c, z^4-c und z^5-c nur eine Menge (M_{z^3-c}, M_{z^4-c} und M_{z^5-c}) erhalten. Die Mengen hängen wie das Apfelmännchen vollständig zusammen, so daß auch weit entfernte Aussiedler durch unendlich dünne Täler mit dem Hauptkörper verbunden sind.

M_{z^3-c} = (c aus C | z_n divergiert nicht)
M_{z^4-c} = (c aus C | z_n divergiert nicht)
M_{z^5-c} = (c aus C | z_n divergiert nicht)

Haben wir ein fertiges Programm für die Berechnung von dem Apfelmännchen, so brau-



4-1-15 Die Menge zu z^3-c



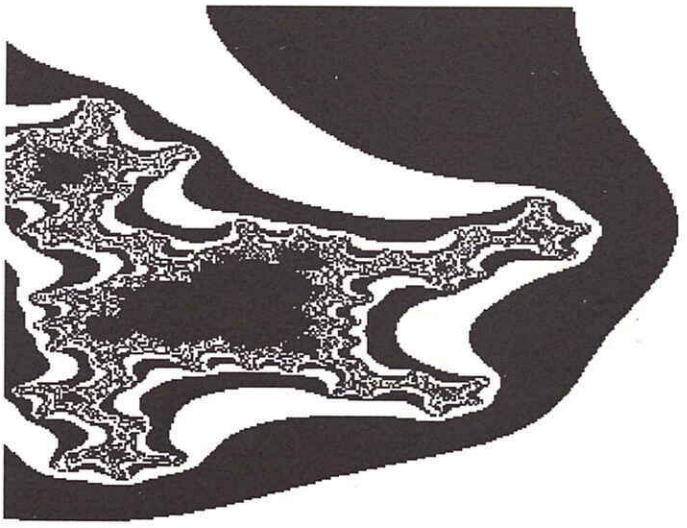
4-1-16 (1) Die Menge z^4-c

chen wir nur die Formel zur Berechnung der neuen Folgenglieder zu modifizieren, da die gleichen Abbruchkriterien verwendet werden. Der Algorithmus zur Entstehung und Berechnung bleibt also gleich.

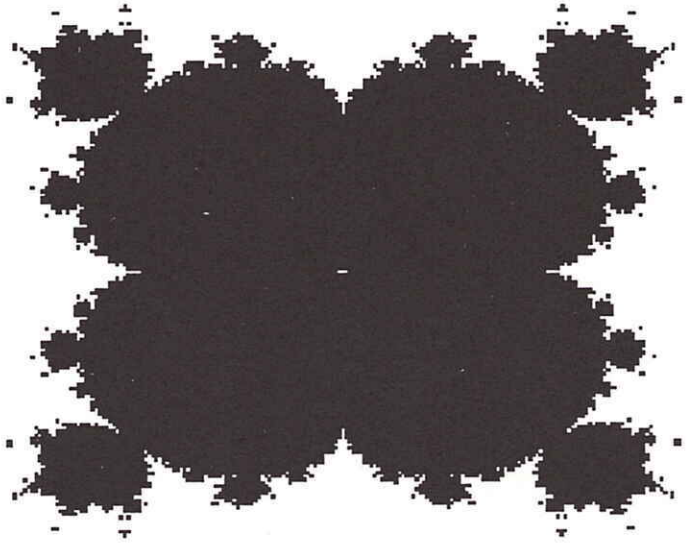
Werten wir nur die Punkte aus, für die die Folge unbestimmtes Verhalten zeigt (n_{max}-Durchläufe werden benötigt), so zeichnen wir die eigentlichen Mengen. Bei Berücksichtigung der benötigten Durchläufe bis zum Abbruch e_{max} (die Folge divergiert!) zeichnen wir Höhenlinien um die Mengen. Dies Vorgehen ist uns aber schon von der Untersuchung von z^2-c bekannt.

Besonders interessant für eine nähere Untersuchung ist nach wie vor der Rand der Menge und speziell weiter entfernte Punkte vom Hauptkörper, die noch zur Menge gehören. Hier finden wir wieder deutliche Beispiele für das Phänomen der Selbstähnlichkeit. Experimentieren Sie selbst! Die Beispiele an Hand der Formeln z^3-c, z^4-c und z^5-c sollen Ihnen zeigen, daß zwar immer Bilder der gleichen Art (Landkarten gleichen sich auf den ersten Blick bei genauerer Betrachtung werden sie erst interessant!) entstehen, aber die Formvielfalt und ihre Variationen endlos sind. Entwickeln und experimentieren Sie

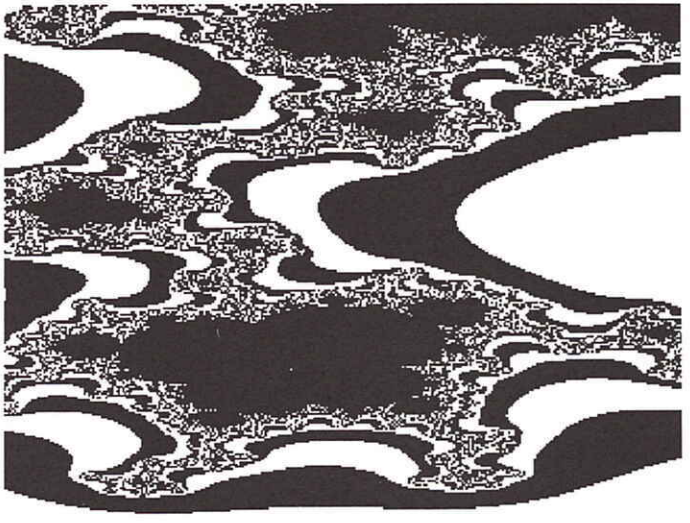
4-1-16 (2) (1) Selbstähnlichkeit überall!



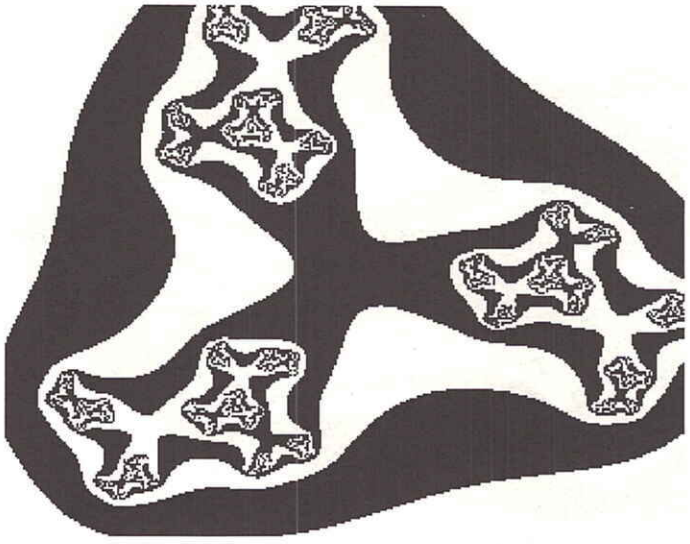
4-1-16 (2) (2) Schwarz-Weiß-Interpretation von Abb. 4-1-16 (2) (1)



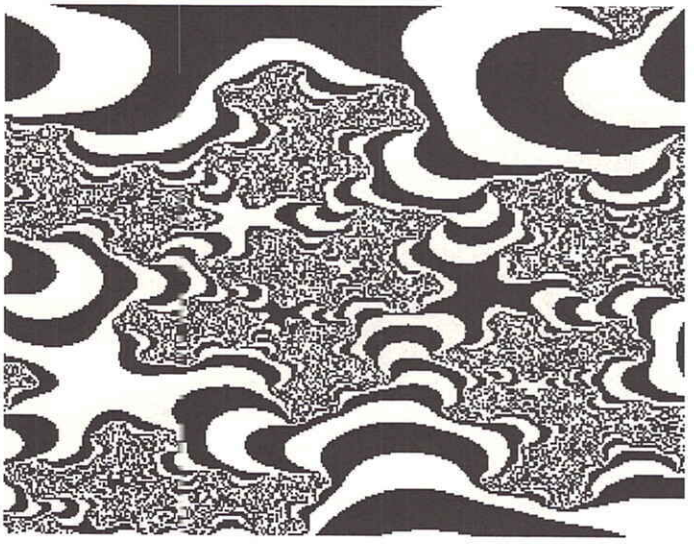
4-1-17 (1) Die fraktale Menge zur Funktion z^5-c



4-1-17 (2) Selbstähnlichkeit (Vergrößerung aus Abbildung 4-1-17 (1))

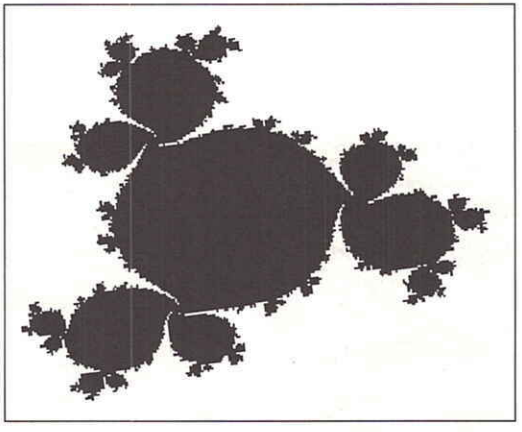


4-1-18 (1) »Julia-Menge« höherer Ordnung



4-1-18 (2)

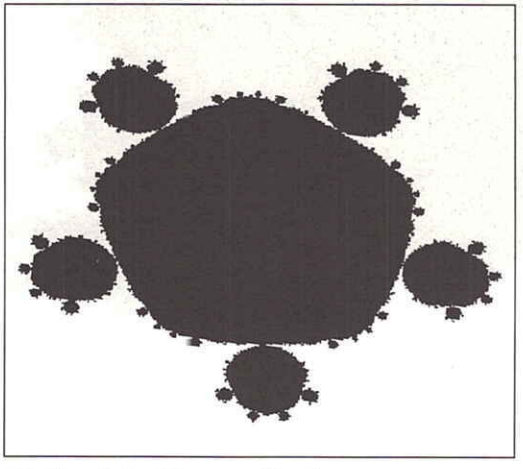
4-1-18 (3) »Julia-Menge« z^3-c



mit eigenen Formeln und Untersuchungskriterien!

Es ist sinnvoll, die Mengen, die wir mit unseren vier Formeln beschreiben, zu vergleichen. Dabei stellen wir folgendes fest: Für z^2-c (Mandelbrot-Menge) und z^4-c, also Formeln mit geradem Exponenten, sind die Mengen zur Reelen-Achse symmetrisch. Sie ähneln sich sehr. Beide haben eine Spitze in die gleiche Richtung und einen bauchigen Hauptkörper. Bei z^4-c scheinen die Knospen wesentlich häufiger aufzutreten.

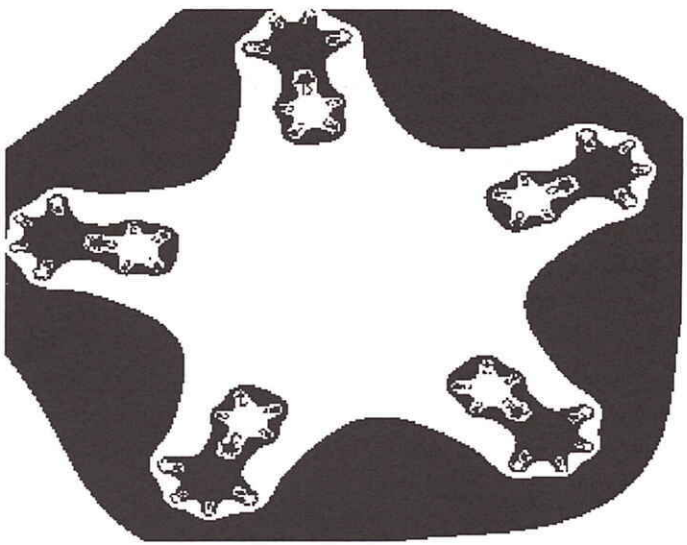
Dagegen erhalten wir für z^3-c und z^5-c, also Formeln mit ungeradem Exponenten, Bilder, die sowohl zur Imaginär-, als auch zur Real-Achse symmetrisch sind.



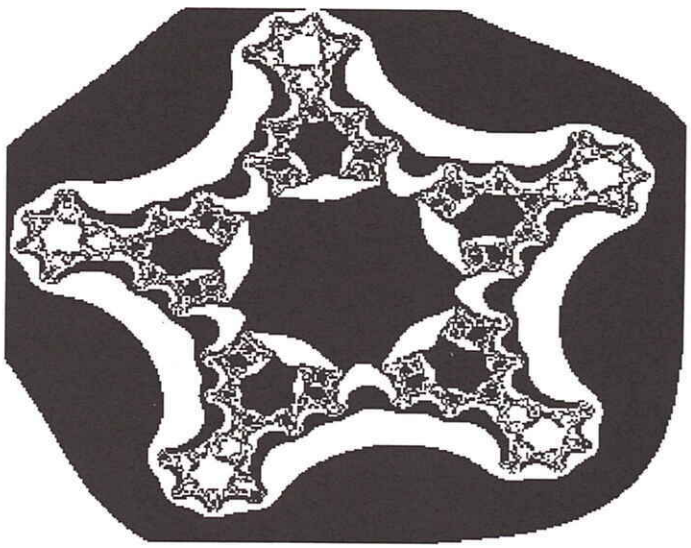
4-1-19 »Julia-Menge« für z^5-c

2) Auswertung analog zu den Julia-Mengen, daß heißt, x_0 wird durch den Bildpunkt bestimmt und c fest gewählt.

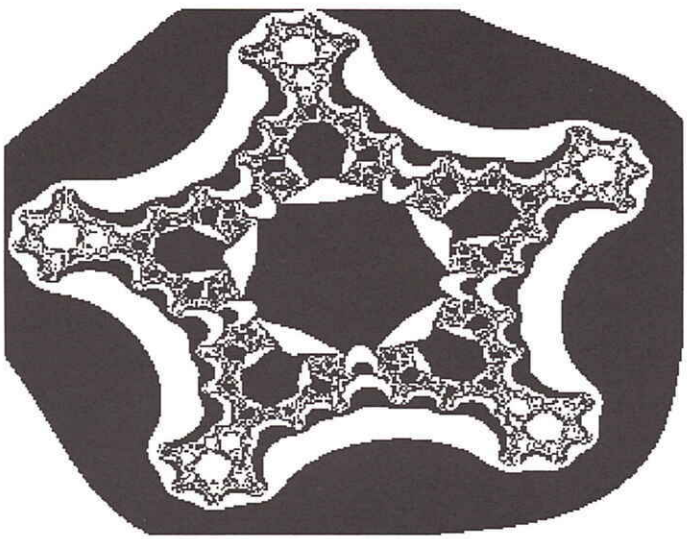
Siehe zur Verdeutlichung dieser Mengen die Abbildungen 4-1-18 und 4-1-19 für z^3-c und z^5-c. Die Bilder belegen, daß die für die Folge z^2-c erkannten Zusammenhänge und Eigenschaften auch für Funktionen höheren Grades erhalten bleiben, zum Beispiel die Selbstähnlichkeit oder der Zusammenhang zwischen Julia- und Mandelbrot-Menge.



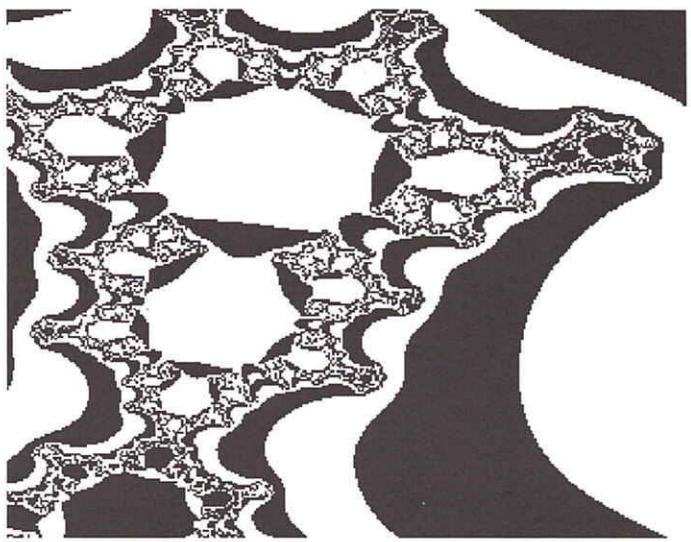
4-1-19 (1)



4-1-19 (2)



4-1-19 (3)



4-1-19 (4)

Wie bei den Julia-Mengen werden wir unendlich viele verschiedene Mengen in Abhängigkeit der Wahl der komplexen Konstanten c entdecken (J_{z^3-c}, J_{z^4-c} und J_{z^5-c}). Wählen wir c aus der zugehörigen Menge, die nach dem Mandelbrot-Verfahren erstellt wird, hängen die Mengen zusammen, ansonsten nicht. Dieses Verhalten kennen wir bereits von den Julia-Mengen. Diese Eigenschaft können wir wieder dazu verwenden, um M_{z^3-c}, M_{z^4-c} und M_{z^5-c} zu beschreiben:

M_{z^3-c} = (c aus C | J_{z^3-c} zusammenhängend)
M_{z^4-c} = (c aus C | J_{z^4-c} zusammenhängend)
M_{z^5-c} = (c aus C | J_{z^5-c} zusammenhängend)

Interessant ist auch hier ein Vergleich der Mengen, die aufgrund der vier verschiedenen Folgen entstehen. Die Julia-Mengen der Folge z^2-c sind zum Ursprung punktsymmetrisch. Sie weisen grundsätzlich eine Orientierung längs einer Geraden auf, besitzen also sozusagen zwei Enden. Bei z^3-c haben die Bilder drei Enden. Sie sind gemäß einem gleichseitigen Dreieck angeordnet. Bei z^4-c erhalten wir analog 4 und bei z^5-c fünf Enden. Die Orientierung der Achsen entspricht den Grenzverläufen der Auswertungen der Formeln z^3-1, z^4-1 und z^5-1 mit dem Newton-Verfahren. Worin besteht der Zusammenhang?

3) Weitere Schnitte

Außer der bisherigen Parameterwahl können wir selbstverständlich wieder Schnitte durch den »vierdimensionalen« Raum legen, die nicht parallel zur c- bzw. z-Ebene liegen. Wie bei z^2-c erhalten wir Verzerrungen der bisherigen Mengen.

Zusammenfassend können wir sagen, die Komplexität der Mengen nimmt mit steigendem Exponent der Entstehungs-Formel zu. Die Erkenntnisse, die wir für die Mandelbrot-Menge und die Julia-Mengen gefunden haben, bleiben erhalten! Bisher haben wir nur »einfache« Formeln verwendet. Es ist aber auch sinnvoll, gemischte Formeln wie z^3-2*z^2-c auszuprobieren!

4.1.9 Zusammenfassung der Formeln Hinweise zur Berechnung von Julia-Mengen, Mandelbrot-Menge und Schnitten

In diesem Fall gestaltet sich die Berechnung erheblich einfacher, da keine komplexen Brüche auftreten, deren Nenner reell gemacht werden muß.

1) Formel: $z_{n+1}=z_n^2-c$

Hier und im weiteren sei $Im(z_n)=x_n$ und $Re(z_n)=y_n$ gesetzt, und die komplexe Zahl c besitze den Realteil cre und den Imaginärteil cim. z_n ergibt sich dann zu:

$Re(z_{n+1})=x_{n+1}=x_n^2-y_n^2-c_{im}$
$Im(z_{n+1})=y_{n+1}=2*x_n*y_n-c_{re}$

2) Formel: $z_{n+1}=z_n^3-c$

$Re(z_{n+1})=x_{n+1}=x_n^3-3*x_n*y_n^2-c_{re}$
$Im(z_{n+1})=y_{n+1}=3*x_n^2*y_n-y_n^3-c_{im}$

3) Formel: $z_{n+1}=z_n^4-c$

Um die Formeln zu vereinfachen, wird es nötig, zwei weitere Hilfsgrößen zu definieren:

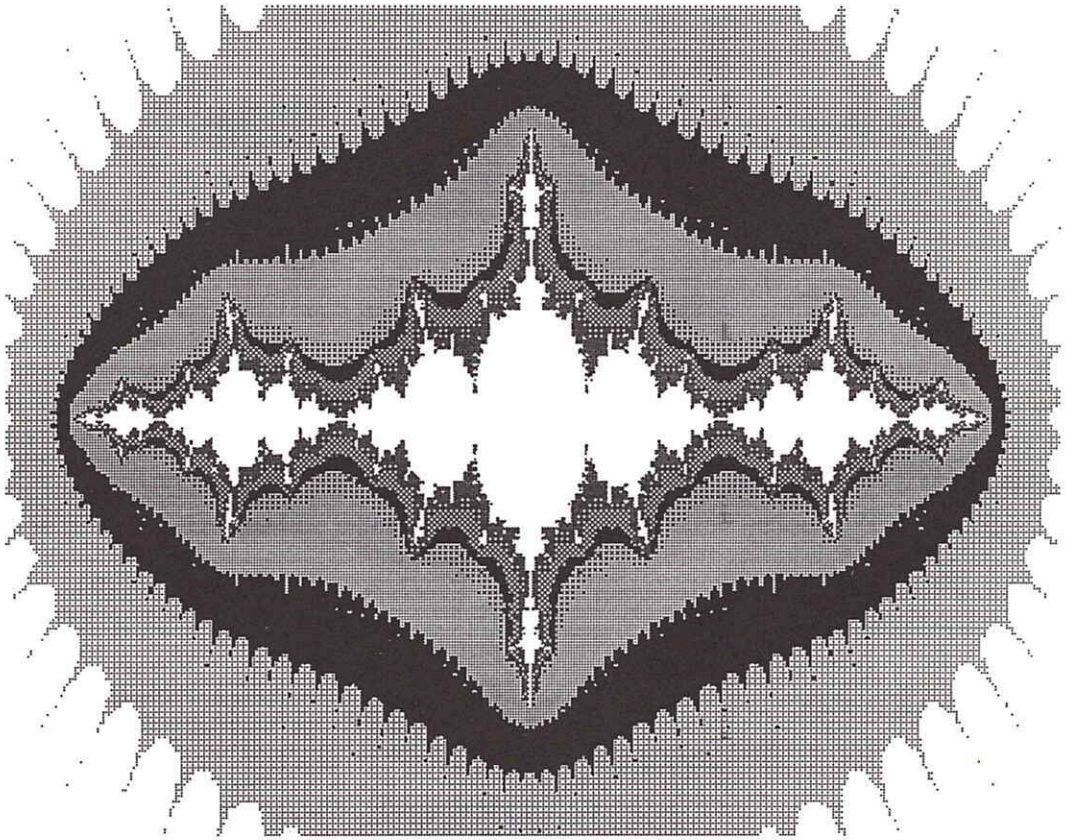
$re=x_n^3-3*x_n*y_n^2$
$im=3*x_n^2*y_n-y_n^3$

Damit ergibt sich das neue Folgenglied zu:

$Re(z_{n+1})=x_{n+1}=re*x_n-im*y_n-c_{re}$
$Im(z_{n+1})=y_{n+1}=re*y_n+im*x_n-c_{im}$

4) Formel: $z_{n+1}=z_n^5-c$

Hier wird es sogar sinnvoll vier Hilfsgrößen einzuführen:



4-1-20 (I) Eine »Julia-Menge« einmal anders ($|y| > e_{max}$)

$re_1=x_n^3-3*x_n*y_n^2$ (siehe Funktion z^4-c)
$im_1=3*x_n^2*y_n-y_n^3$

und

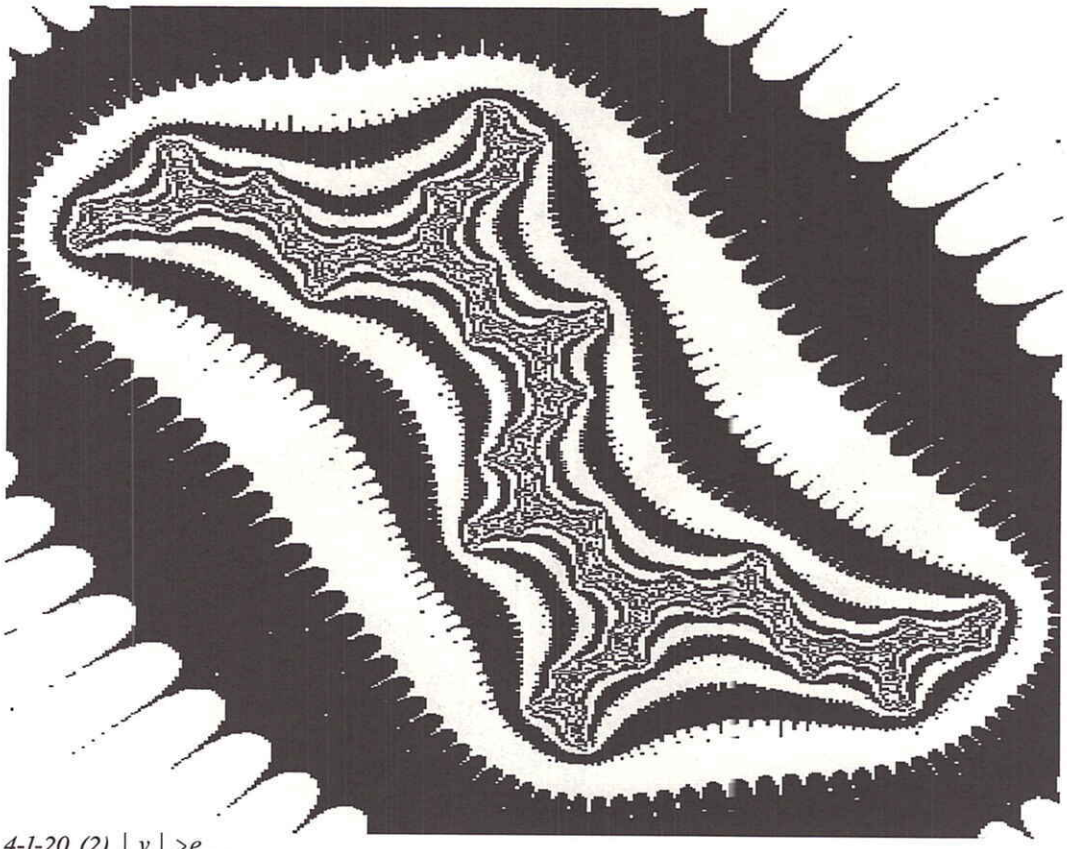
$im_2=re_1*y_n+im_1*x_n$
$re_2=re_1*x_n-im_1*y_n$

Damit ergibt sich das neue Folgenglied zu:

$Re(z_{n+1})=x_{n+1}=re_2*x_n-im_2*y_n-c_{re}$
$Im(z_{n+1})=y_{n+1}=re_2*x_n+im_2*y_n-c_{im}$

4.1.10 Anmerkung zu Berechnungen bei komplexen Mengen

Die Rechenzeit vieler Grafiken ist nicht ganz unerheblich. Gerade wenn man darauf wartet, daß ein Bild fertig wird, scheint sich die Berechnung endlos zu erstrecken. Und weggehen und den Rechner alleine arbeiten lassen will man auch nicht so recht, denn dann würde er einige Zeit völlig unnötig laufen. Also was tun? Eine Möglichkeit wäre es, das Bild während der Berechnung speichern zu können und dann nach dem Wiedereinladen



4-1-20 (2) $|y| > e_{max}$

weiterberechnen zu lassen. Dies ist beim Fraktal-Generator möglich. Noch komfortabler wäre es allerdings, dem Programm eine Liste von Koordinaten übergeben zu können, damit diese dann automatisch nacheinander berechnet und speichert. Schreiben Sie doch ein solches Programm!

Um die Berechnung zu beschleunigen, ist es beim Amiga sinnvoll, einen Screen mit möglichst wenigen Bitplanes im Vordergrund zu haben, denn die Darstellung eines Screens mit mehreren Bitplanes bremst die CPU erheblich! Deshalb sollte der Screen, auf dem gezeichnet wird durch Drücken der linken Amiga-Taste und »N« nach hinten gelegt werden. Will man das Bild wieder sehen, so holt man durch Drücken der linken Amiga-Taste und »M« den Screen wieder nach vorn.

4.2 Von der Ebene zur Landschaft – dreidimensionale Untersuchungen der Mandelbrot-Menge

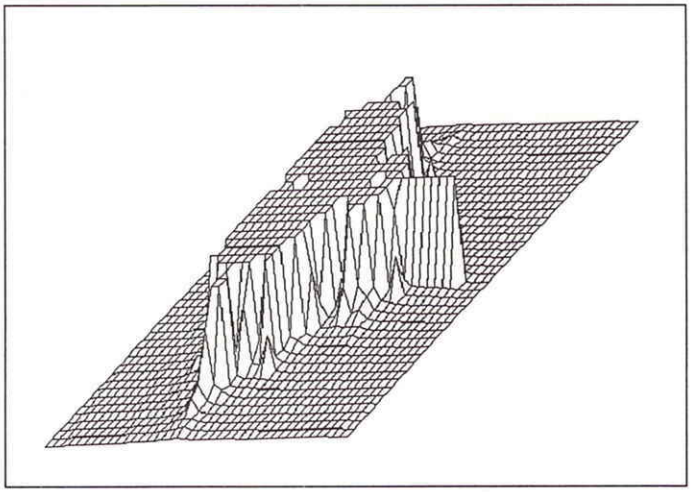
Bisher haben wir nur zweidimensionale Bilder berechnet. Die Zuordnung der Farben geschah mit Hilfe der Modulo-Funktion aus den benötigten Durchläufen. Wir haben bereits damals erkannt, daß die bei dieser Auswertung entstehenden Bereiche gleicher Farbe so etwas wie Höhenlinien sind. Haben wir also eine Draufsicht auf eine dreidimensionale Landschaft vor uns? Waren unsere bisher berechneten Bilder Landkarten der Mandelbrot-Landschaft?

Beide Fragen sind mit »Ja« zu beantworten. Denn interpretieren wir die notwendigen Durchläufe bis zum Abbruch als Höhe einer Landschaft, so erhalten wir ein »Konvergenzgebirge«. Aus diesem Gebirge können wir aufgrund der unterschiedlichen Höhen erkennen, wie lange die Folge bis zur Divergenz bzw. Konvergenz braucht. Dies gelingt uns jetzt genauer als bei der zweidimensionalen Auswertung, da wir hier nicht nur eine Auflösung von 32 bzw. 64 aufgrund der beschränkten Anzahl der Farben erreichen können.

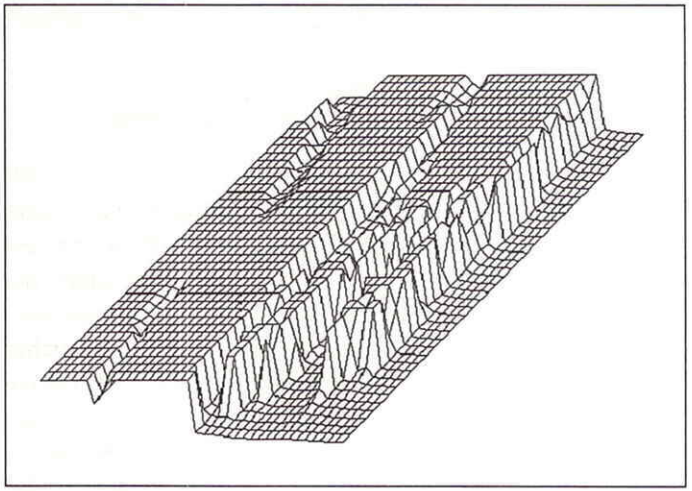
Das Zeichnen eines solchen dreidimensionalen Gebirges kann auf verschiedene Weisen geschehen. Entweder wir verwenden die Bildschirmrasterung oder legen uns in der z-Ebene ein Raster an. Beide Methoden sind im Kapitel über die dreidimensionale Grafik erklärt.

Das folgende Programm verwendet die zweite Methode. Ein beim Starten des Programms anzugebender Bereich wird in ein Raster zerlegt. Die Rasterpunkte dienen als Stützpunkte für die Landschaft. Ihnen wird die Anzahl der notwendigen Durchläufe `n` als Höhe zugeordnet. Die Ausgabe erfolgt dann durch Projektion des Rasters auf die Bildschirmebene. Die verdeckten Linien werden dadurch unterdrückt, daß die Ausgabe von der am weitesten entfernten Fläche beginnt. Die Flächen werden in Hintergrundfarbe gezeichnet und eingerahmt. Als Ergebnis erhalten wir dann ein Gittermodell des Apfelmännchen-Ausschnitts. Diese Art der Auswertung ist sehr anschaulich, hat jedoch zwei Nachteile: Erstens ist das erhaltene Bild einfarbig und zweitens reduzieren wir die Auflösung im Vergleich zur zweidimensionalen Ausgabe erheblich. Hatten wir bei einem LoRes-Bild 320*250=8000 Bildpunkte, so beschränken wir uns jetzt zum Beispiel auf 20*20=400 Stützpunkte!

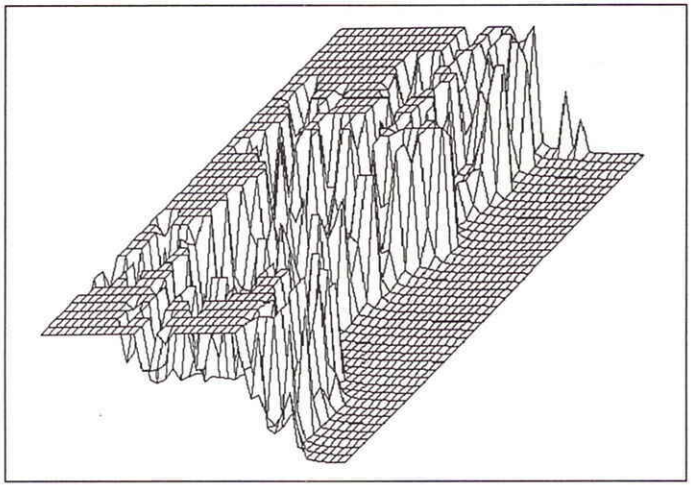
Den mangelnden Farben kann abgeholfen werden. Wie bereits beschrieben, kann mit Hilfe eines Beleuchtungsvektors jeder Fläche eine Farbe zugeordnet werden (siehe Programm *apfel3d2.c* auf der Diskette). Auch Perspektive kann bei der Ausgabe benutzt werden, um realistische Bilder zu erhalten!
Wenn man die mit diesem Programmm berechneten Bilder betrachtet, liegt der Gedanke nahe, einen Flug durchs Apfelmännchen zu realisieren. Aber da gibt es einige Probleme: Die Berechnung der Höhen über die Stützpunkte dauert verständlicherweise zu lange, als daß eine Echtzeitberechnung für einen kontinuierlichen Flug möglich wäre. Also müßten wir zuerst die einzelnen Bilder berechnen und dann als Flug abspielen. Eine solche Sequenz könnte mit dem Film-Projektor *pro.c* vorgeführt werden. Doch auch das dabei erhaltene Resultat ist recht unbefriedigend. Wird die Schrittweite bei der Berechnung zu groß gewählt, ändert sich die Landschaft zu schnell. Bei einer ungleichmäßigen Veränderung in x- und y-Richtung kann es



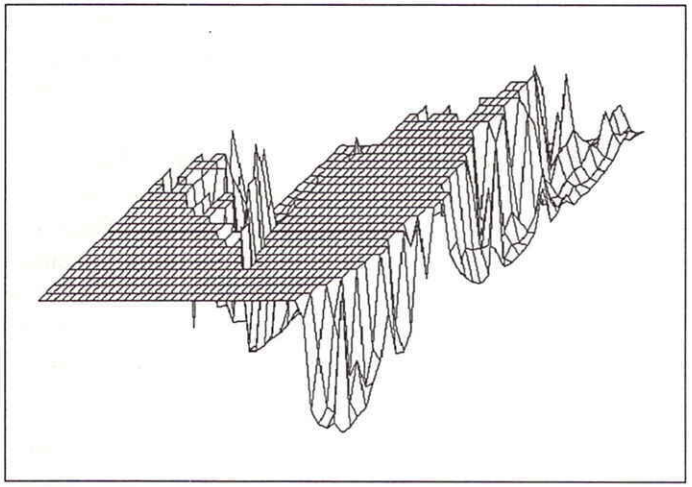
4-2-1



4-2-2



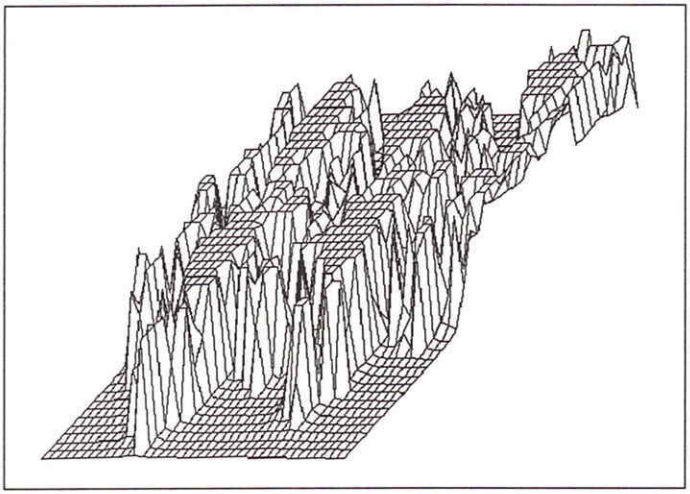
4-2-5



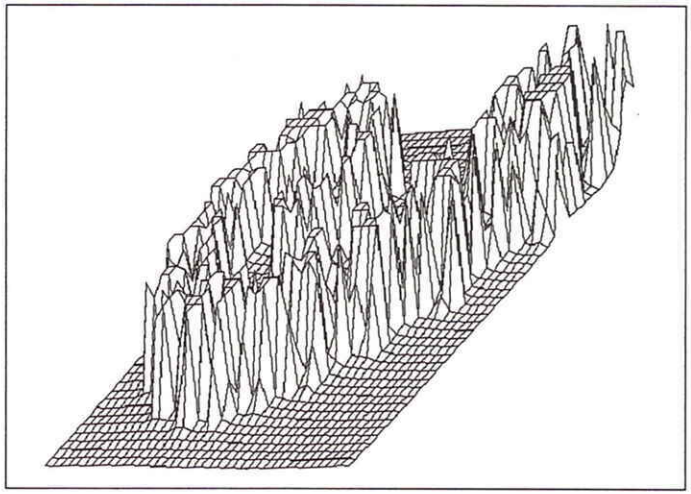
4-2-6

Die Mandelbrot-Menge als Landschaft

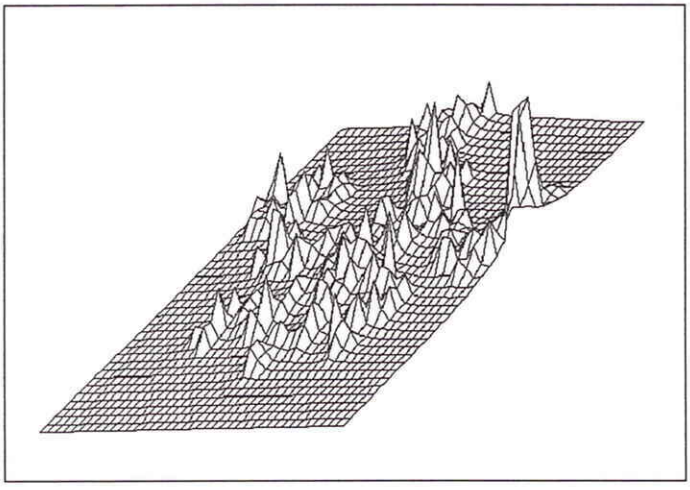
dabei zu Verzerrungen kommen. Aber selbst dann, wenn wir diese beiden Erkenntnisse beachten, werden wir nicht zu dem gewünschten Ergebnis kommen! Im Randbereich (und gerade das ist der Reiz des fraktalen Randes!) kommt es zu sehr starken Schwankungen der Höhe, wenn wir einen Stützpunkt nur ein wenig verschieben. Im Film wirkt deshalb der Randbereich der Menge sehr unruhig. Dauernd tauchen hohe



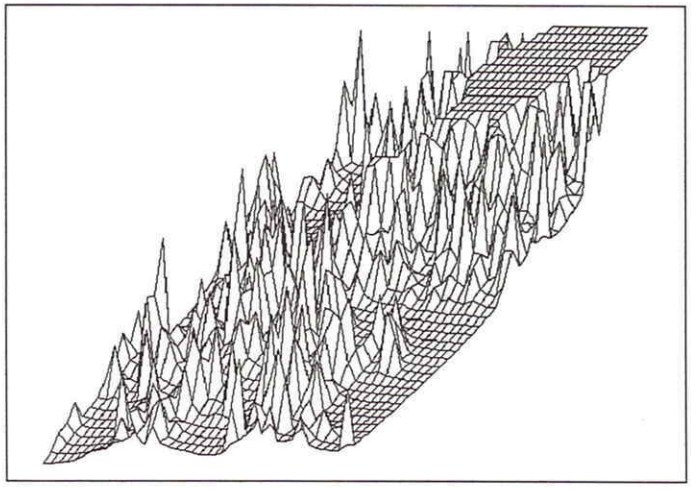
4-2-3



4-2-4



4-2-7



4-2-8

Spitzen auf, die dann aber sofort wieder verschwinden. Es ist klar, daß ein gesteuerter (interaktiver) Flug noch mehr Probleme aufwerfen wird!

Programm: *apfel3d.c*
(siehe Abbildungen 4-2-1 bis 4-2-8)

```
/*************************************************************************/
/* apfel3d.c                                                  28.12.87   */
/* Version 1.Ø                                                           */
/*                                                                       */
/*                                                                       */
/*                                                                       */
/* fuer Markt+Technik geschrieben von Heiko Knappe                       */
/*                                                                       */
/* Komplexe Dynamische Systeme                                           */
/*************************************************************************/


#include <exec/types.h>
#include <math.h>
#include <intuition/intuition.h>
#include <graphics/gfxmacros.h>

#define WIDTH   64Ø  /* Screen Modus */
#define HEIGHT  5ØØ
#define DEPTH   1
#define FLAGS   HIRES+LACE
#define FWIDTH  3ØØ
#define FHEIGHT 3ØØ
#define FXOFF   32Ø
#define FYOFF   15Ø

#define N 4Ø         /* Anzahl der Stuetzpunkte in x,y Richtung */

/* falls das Bild gespeichert werden soll! */
/* extern short save(); */

struct IntuitionBase *IntuitionBase;
struct GfxBase *GfxBase;
struct Screen *MyScreen;
struct RastPort *MyRastPort;
struct AreaInfo MyAreaInfo;
struct TmpRas MyTmpRas;
struct NewScreen MyNewScreen=
 {Ø,Ø,WIDTH,HEIGHT,DEPTH,1,Ø,FLAGS,CUSTOMSCREEN,Ø,Ø,Ø,Ø};

int   n,nmax=1ØØ;          /* Iterationsschritte                */
float emin=.ØØ1,emax=1ØØØØØ; /* Abbruchkriterium                */
float rmin=-1.5,rmax=2.5;   /* Grenzen des Bildes                */
float imin=-1.2,imax=1.2;
float rst,ist;
float re,im,re1,re2,im1,im2,cre,cim;
                            /* re,im laufender Re- und Im-Teil   */
                            /* re1,re2,im1,im2 → Folgenglieder   */
float d;                    /* Abstand zu dem Bezugspunkt        */
int   i,j,k;
short land[N][N];           /* Feld mit den Punktwerten          */
short x[4],y[4];            /* 4 Eckpunkte der Flaechen          */
short stx,sty;
UWORD buffer[5*1Ø];         /* Variablen fuer AreaFill           */
long  *sp;


/***********************/
/* Hauptprogramm       */
/***********************/

main(argc,argv)
int argc;
char *argv[];
{
 if((argc!=2)&&(argc!=1))
 {
    printf("Eingabe: %s <nmax,rmin,rmax,imin,imax>\n",argv[Ø]);
    printf(" nmax: maximale Anzahl der Duerchlaeufe\n");
    printf(" [rmin,rmax] [imin,imax]: Wertebereich der Bildebene\n");
    close_things();
 }
 if(argc==2)
    if(sscanf(argv[1],"%d,%f,%f,%f,%f",&nmax,&rmin,&rmax,&imin,&imax)!=5)
    {
       printf("falsche Eingabe\n");
       close_things();
    }
 printf("\nParameter:\n emin=%f emax=%f\n",emin,emax);
 printf(" nmax=%d\n [%f,%f] [%f,%f]\n",nmax,rmin,rmax,imin,imax);
 emin*=emin;  /* mit dem Quadrat zu vergleichen erspart Wurzeln zu */
 emax*=emax;  /* berechnen → Schnelligkeit                         */
```

```
 83  apfel();
 84  open_things();
 85  ausgabe();
 86  while(MyScreen->TopEdge<1ØØ);
 87  /* falls das Bild gespeichert werden soll      */
 88  /* save("Apfel.pic",1,MyScreen);               */
 89  close_things();
 9Ø }
 91
 92
 93 /***********************/
 94 /* Initialisierung     */
 95 /***********************/
 96
 97 open_things()
 98 {
 99  if((IntuitionBase=OpenLibrary("intuition.library",Ø))==Ø)
1ØØ     close_things();
1Ø1  if((GfxBase=OpenLibrary("graphics.library",Ø))==Ø)
1Ø2     close_things();
1Ø3  if((MyScreen=OpenScreen(&MyNewScreen))==Ø)
1Ø4     close_things();
1Ø5  MyRastPort=&(MyScreen->RastPort);
1Ø6  init_colors();
1Ø7  init_fill();
1Ø8 }
1Ø9
11Ø init_fill()
111 {
112  if((sp=AllocRaster(WIDTH,HEIGHT))==Ø)
113     close_things();
114  MyRastPort->AreaInfo=&MyAreaInfo;
115  MyRastPort->TmpRas=InitTmpRas(&MyTmpRas,sp,RASSIZE(WIDTH,HEIGHT));
116  InitArea(&MyAreaInfo,&buffer[Ø],1Ø);
117 }
118
119 close_things()
12Ø {
121  if(sp!=Ø) FreeRaster(sp,64Ø,25Ø);
122  if(IntuitionBase!=Ø) CloseLibrary(IntuitionBase);
123  if(GfxBase!=Ø) CloseLibrary(GfxBase);
124  if(MyScreen!=Ø) CloseScreen(MyScreen);
125  exit();
126 }
127
128 init_colors()
129 {
13Ø  SetRGB4(&(MyScreen->ViewPort),Ø,Ø,Ø,Ø);
131  SetRGB4(&(MyScreen->ViewPort),1,15,15,15);
132 }
133
134
135 /***********************/
136 /* Berechnung der      */
137 /* Stuetzpunkte        */
138 /***********************/
139
14Ø apfel()
141 {
142  printf("Berechnung der Stuetzpunkte \n");
143  rst=(rmax-rmin)/N;
144  ist=(imax-imin)/N;
145  for(i=Ø;i<N;++i) /* in x-Richtung */
146  {
147     re=rmin+i*rst;
148     for(j=Ø;j<N;++j) /* in y-Richtung */
149     {
15Ø        im=imin+j*ist;
151        for(n=re1=im1=Ø,cre=re,cim=im;n<nmax;++n)
152        {
153           im2=2*re1*im1-cim,re2=re1*re1-im1*im1-cre;
154           im1=im2,re1=re2;
155           d=im1*im1+re1*re1;
156           /* Ist der Attraktor erreicht? */
157           if(d<emin)  break;
158           /* Attraktor Unendlich?        */
159           if(d>emax)  break;
16Ø        }
161        land[i][j]=n;
162     }
163  }
164  printf("-> fertig\n");
165 }
```

```
166
167
168 /************************/
169 /* Ausgabe              */
170 /************************/
171
172 ausgabe()
173 {
174  stx=FWIDTH/N,sty=FHEIGHT/N;
175  for(i=0;i<N-1;++i)
176  {
177      for(j=0;j<N-1;++j)
178      {
179         /* Berechnung der Eckpunkte der Flaechen */
180         x[0]=FXOFF+stx*i-sty*j,y[0]=FYOFF-land[i][j]+sty*j;
181         x[1]=FXOFF+stx*(i+1)-sty*j,y[1]=FYOFF-land[i+1][j]+sty*j;
182         x[2]=FXOFF+stx*(i+1)-sty*(j+1),y[2]=FYOFF-
          land[i+1][j+1]+sty*(j+1);
183         x[3]=FXOFF+stx*i-sty*(j+1),y[3]=FYOFF-land[i][j+1]+sty*(j+1);
184         for(k=0;k<3;++k)
185            if(y[k]<0) y[k]=0;
186         zeich();
187      }
188  }
189 }
190
191 zeich()
192 {
193  SetAPen(MyRastPort,0);  /* Flaeche / Umgrenzung zeichnen */
194  SetOPen(MyRastPort,1);
195  AreaMove(MyRastPort,x[0],y[0]); AreaDraw(MyRastPort,x[1],y[1]);
196  AreaDraw(MyRastPort,x[2],y[2]); AreaDraw(MyRastPort,x[3],y[3]);
197  AreaDraw(MyRastPort,x[0],y[0]); AreaEnd(MyRastPort);
198 }
```

Auf die an anderer Stelle erwähnte Pseudo-3-D-Grafik-Methode möchte ich hier nicht weiter eingehen. Sie wird im Programm *fraktal.c* (Fraktal-Generator) verwendet!

Wir können auch das Programm *3-d.c* benutzen, um die Mandelbrot-Menge räumlich darzustellen. Wie gehabt, berechnen wir unsere Stützpunkte über die komplexe Ebene. Das Ergebnis geben wir dann aber nicht mehr als Bild aus, sondern legen es in einem »*.dat*«-File ab, das das Programm *3-d.c* einlesen kann. Zuerst werden die Koordinaten aller Punkte berechnet (bezogen auf einen Bereich von `[-1,1][-1,1][-1,1]`) und abgespeichert. Danach speichern wir die Flächendefinitionen ab. Wir erhalten ein Text-File (das wir auch editieren können) der folgenden Form:

```
P x,y,z
P ...

F 1,2,12
 ...
```

Das folgende Programm übernimmt die Berechnung und Speicherung in der angesprochenen Art und Weise. Das »*.dat*«-File besitzt den Namen *mandel.dat*. Damit startet man das Programm *3-d.c* wie folgt:

```
3-d.run Mandel
```

Jetzt können wir sogar die räumliche Landschaft drehen und im Raum verschieben. Die Ausgabe hat jedoch einige Nachteile: Erstens werden auch die verdeckten Linien gezeichnet, so daß das Bild schwierig zu erkennen ist. Außerdem schränken wir die Unterteilung des Gitters noch weiter ein, da das 3-D-Programm nur eine begrenzte Menge von Punkt- und Flächendefinitionen bewältigt. Leider ist bei der großen Anzahl der zu zeichnenden Linien die Animation-Funktion sehr langsam, so daß keine fortlaufende Bewegung entsteht.

Programm: *3-d-generator.c*

```
/*************************************************************************/
/* 3-d-generator.c                                                       */
/* Version 1.Ø                                                           */
/*                                                                       */
/* Erzeugt ein von dem Programm 3-d.c weiterverarbeitbares File.         */
/* Dieses File Mandel.dat wird dann wie folgt verwendet:                 */
/*  3-d.run Mandel                                                       */
/*                                                                       */
/* fuer Markt+Technik geschrieben von Heiko Knappe                       */
/*                                                                       */
/* Komplexe Dynamische Systeme                                           */
/*************************************************************************/


#include <exec/types.h>
#include <math.h>
#include <stdio.h>
#include <libraries/dos.h>

#define N 1Ø            /* Anzahl der Stuetzpunkte in x,y Richtung */
#define XRANGE .8
#define XOFF.4
#define YRANGE .8
#define YOFF.4
#define ZRANGE .8
#define ZOFF.4


int   n,nmax=1ØØ;           /* Iterationsschritte                 */
float emin=.ØØ1,emax=1ØØØØØ; /* Abbruchkriterium                   */
float rmin=-1.5,rmax=2.5;    /* Grenzen des Bildes                 */
float imin=-1.2,imax=1.2;
float rst,ist;
float re,im,re1,re2,im1,im2,cre,cim;
                             /* re,im laufender Re- und Im-Teil    */
                             /* re1,re2,im1,im2  -> Folgenglieder  */
float d;                     /* Abstand zu dem Bezugspunkt         */
int   i,j,k;
short land[N][N];            /* Feld mit den Punktwerten           */
char  name[]="Mandel.dat";
FILE  *file;


/***********************/
/* Hauptprogramm       */
/***********************/

main(argc,argv)
int argc;
char *argv[];
{
 if((argc!=2)&&(argc!=1))
 {
    printf("Eingabe: %s <nmax,rmin,rmax,imin,imax>\n",argv[Ø]);
    printf(" nmax: maximale Anzahl der Duerchlaeufe\n");
    printf(" [rmin,rmax] [imin,imax]: Wertebereich der Bildebene\n");
    exit(FALSE);
 }
 if(argc==2)
    if(sscanf(argv[1],"%d,%f,%f,%f,%f",&nmax,&rmin,&rmax,&imin,&imax)
!=5)
    {
       printf("falsche Eingabe\n");
       exit(FALSE);
    }
 printf("\nParameter:\n emin=%f emax=%f\n",emin,emax);
 printf(" nmax=%d\n [%f,%f] [%f,%f]\n",nmax,rmin,rmax,imin,imax);
 emin*=emin;  /* mit dem Quadrat zu vergleichen erspart Wurzeln zu */
 emax*=emax;  /* berechnen -> Schnelligkeit                        */
 apfel();
 ausgabe();
}


/***********************/
/* Berechnung der      */
/* Stuetzpunkte        */
/***********************/

apfel()
{
 printf("Berechnung der Stuetzpunkte \n");
 rst=(rmax-rmin)/N;
```

```
ist=(imax-imin)/N;
for(i=Ø;i<N;++i) /* in x-Richtung */
{
   re=rmin+i*rst;
   for(j=Ø;j<N;++j) /* in y-Richtung */
   {
      im=imin+j*ist;
      for(n=re1=im1=Ø,cre=re,cim=im;n<nmax;++n)
      {
         im2=2*re1*im1-cim,re2=re1*re1-im1*im1-cre;
         im1=im2,re1=re2;
         d=im1*im1+re1*re1;
         /* Ist der Attraktor erreicht?                    */
         if(d<emin)  break;
         /* Attraktor Unendlich?                           */
         if(d>emax)  break;
      }
      land[i][j]=n;
   }
}
printf("-> fertig\n");
}


/***********************/
/* Ausgabe             */
/***********************/

ausgabe()
{
 float fx,fy,fz;

 if((file=fopen(name,"w"))==Ø)
 {
    printf("Daten-File kann nicht geoeffnet werden!!\n");
    exit(FALSE);
 }
 /* Punkte abspeichern */
 for(i=Ø;i<N;++i)
 {
    for(j=Ø;j<N;++j)
    {
       fx=-.5+(float)1/(float)(N-1)*(float)i;
       fz=-.5+(float)1/(float)(N-1)*(float)j;
       fy=-.5+(float)1/(float)nmax*(float)land[i][j];
       fprintf(file,"P %f,%f,%f\n",fx,fy,fz);
    }
    fprintf(file,"\n");
 }
 /* Flaechen abspeichern */
 for(i=1;i<N;++i)
 {
    for(j=1;j<N;++j)
    {
       fprintf(file,"F %d,%d,%d\n",j+(i-1)*N,j+(i-1)*N+1,j+i*N+1);
       fprintf(file,"F %d,%d,%d\n",j+(i-1)*N+1,j+i*N+1,j+i*N);
    }
 }
 fclose(file);
}
```

4.3 Das Newton-Verfahren – Nullstellensuche einmal ganz anders

Sicher weiß jeder, was eine Nullstelle einer Funktion ist und damit auch um die Probleme, diese zu finden.

Hat man eine Funktion `f(x)` gegeben und sucht ihre Nullstellen, also die Lösung der Gleichung `f(x)=0`, so ist es naheliegend, die Gleichung nach dem Parameter `x` aufzulösen. Dies ist jedoch nicht immer möglich. Polynome höheren Grades als vier sind zum Beispiel nicht mehr explizit auflösbar, abgesehen davon, daß der Aufwand sehr schnell anwächst. Deshalb greift man gerne zu Näherungsverfahren und damit zu Rechnern, die zwar die Ergebnisse der zu lösenden Gleichung nicht genau, doch recht schnell und einfach ermitteln können. Diese Näherungs-

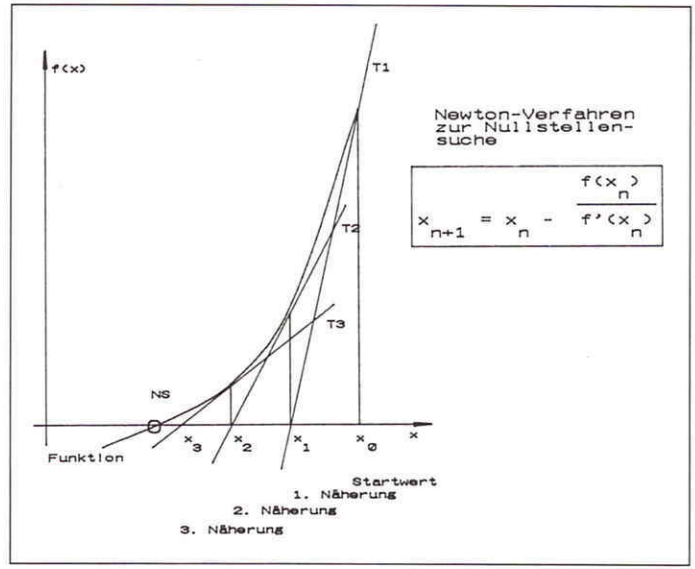
verfahren haben aber auch einen Nachteil, je nach Methode und Startwert führen sie nicht unbedingt zu einem richtigen Ergebnis.

Das Verfahren, das wir hier an dieser Stelle untersuchen wollen, nennt sich Newton-Verfahren. Wieder begegnet uns ein alter Bekannter: die rekursive Folge. Wie wir schon erwarten können, werden also auch hier Probleme auftreten, die es sicher wert sind, untersucht zu werden, wie im folgenden noch deutlich wird.

Gehen wir von einer Funktion f(x) aus (x eine reelle Zahl), so wird die Nullstelle auf folgende Weise angenähert:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Diese Folge wird mit einem Anfangswert $x_\emptyset$ gestartet, der auschlaggebend ist, ob und welche Nullstelle überhaupt gefunden wird. Solange x eine reelle Zahl ist, können wir uns das Vorgehen relativ einfach veranschaulichen. An eine Kurve werden immer wieder Tangenten angelegt, deren Schnittpunkt mit der x-Achse wieder einen neuen Punkt für eine Tangente ergibt. So nähert man sich langsam der Nullstelle der Funktion (siehe Abbildung 4-3-1). Nun ist es auch einfach, ein Beispiel zu finden, bei dem dies nicht der Fall ist. Wie gesagt, ob und welche Nullstelle gefunden wird, hängt immer vom Anfangswert der Folge ab. Bereits hier treten also schon Bereiche auf, in denen die Folge gegen eine Nullstelle konvergiert bzw. divergiert. Natürlich brauchen wir uns nicht nur auf reelle Zahlen zu beschränken. Warum sollten wir nicht komplexe Zahlen verwenden, denn hier werden sich die Konvergenz- und Übergangsbereiche sicher noch interessanter gestalten.



4-3-1

Als erste Funktion wählen wir eine, deren Nullstellen uns bekannt sind:

$f(z) = z^3-1$, wobei z eine komplexe Zahl ist.

Verwenden wir nun das Newton-Verfahren zur Nullstellen-Suche, so müssen wir die Ableitung der Funktion berechnen. (Eine Ableitung ist die Steigung der Tangente in einem bestimmten Punkt.) In unserem Fall:
$f'(z) = 3*z^2$.

Für die Folge ergibt sich also die Gleichung

$$z_{n+1} = z_n - \frac{f(z_n)}{f'(z_n)} = z_n - \frac{z_n^3-1}{3*z_n^2}$$

$$= \frac{2}{3} * z_n + \frac{1}{(3*z_n^2)}$$

Da z_n eine komplexe Zahl ist, muß der Nenner des Terms $\frac{1}{(3*z_n^2)}$ reell gemacht werden, um z_{n+1} als komplexe Zahl darstellen zu können. Dazu muß z_n getrennt in

Real- und Imaginärteil dargestellt werden (Re=Re(z_n) und Im=Im(z_n)). Damit ergibt sich die Folge

$$z_{n+1} = \frac{1}{3} * \left(2*Re+2*Im*i + \frac{(Re^2-Im^2) - 2*i*Re*Im}{(Re^2-Im^2)^2 + 4*Re^2*Im^2}\right),$$

oder getrennt in Imaginär- und Realteil

$$Re(z_{n+1}) = \frac{1}{3} * \left(2*Re + \frac{Re^2-Im^2}{(Re^2-Im^2)^2+4*Re^2*Im^2)}\right)$$

und

$$Im(z_{n+1}) = \frac{1}{3} * \left(2*Im - \frac{2*Re*Im}{(Re^2-Im^2)^2+4*Re^2*Im^2)}\right)$$

Mit diesen Formeln können wir bereits ein Programm schreiben mit dessen Hilfe Bilder berechnet werden, die uns Aussagen über Konvergenz- und Divergenzbereiche der rekursiven Reihe liefern. Wie bereits bei der Auswertung komplexer Folgen angesprochen, geben wir uns drei Werte vor, die uns das Konvergenz-, das Divergenz- und das Unbestimmtheitskriterium angeben: e_{min}, e_{max}, n_{max}. Abgebrochen wird die Entscheidung, ob die Folge für einen bestimmten Anfangswert, und damit für einen Punkt aus der komplexen Ebene bzw. einen Bildpunkt entweder konvergiert, divergiert oder unbestimmt ist, je nach Abstand des neuen Folgenglieds zu allen Nullstellen (Attraktoren!). Ist der Abstand zu einer Nullstelle kleiner als e_{min}, so konvergiert die Folge gegen diese, ist der Abstand größer e_{max}, so divergiert sie.

Wichtig zu wissen ist natürlich an dieser Stelle, wie man den Abstand d zweier komplexer Zahlen z_1 und z_2 berechnet:

$$d = |z_2-z_1| = |z_1-z_2| =$$
$$= |(Re(z_1)-Re(z_2)) + i*(Im(z_1)-Im(z_2))|$$
$$= \sqrt{((Re(z_1)-Re(z_2))^2+(Im(z_1)-Im(z_2))^2}$$

In der Praxis ist es sinnvoll, mit dem Quadrat des Abstands, also d^2, zu arbeiten, um so die Rechenzeit für das Ausrechnen der Wurzel zu sparen. Damit muß natürlich auch mit den Quadraten von e_{min} und e_{max} verglichen werden.

Unsere Funktion $f(z)=z^3-1$ hat Nullstellen für Lösungen der Gleichung

$$z^3-1=0 \Rightarrow z^3=1 \Rightarrow z = \sqrt[3]{1}$$

Rechnet man mit reellen Zahlen, so ist also 1 eine dreifache Nullstelle der Funktion. Bei komplexen Zahlen gibt es aber außer 1 zwei komplexe Werte, die die Gleichung erfüllen und nicht vergessen werden dürfen. In unserem Fall erhalten wir die Nullstellen:

$$z_1=1 \ , \ z_2=.5*(-1+\sqrt{3}*i) \ ,$$
$$z_3=.5*(-1-\sqrt{3}*i)$$

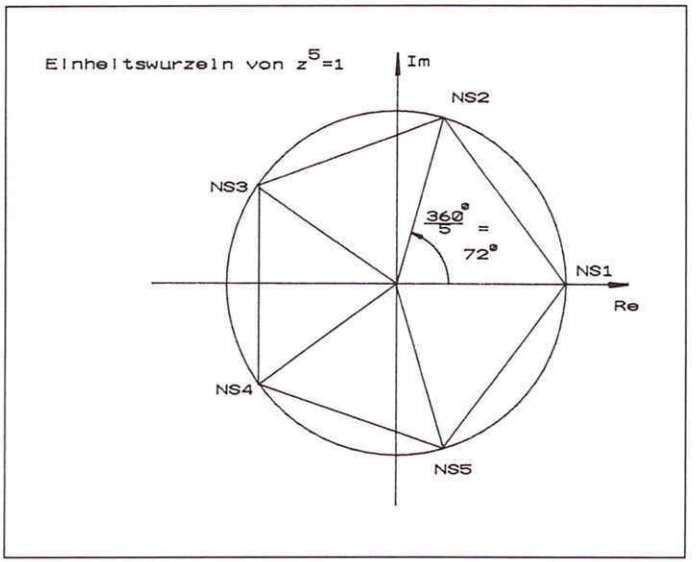
Diese drei Lösungen liegen auf einem Kreis mit dem Radius 1, dem Einheitskreis, um den Ursprung und bilden ein gleichseitiges Dreieck. Man spricht deshalb auch von Einheitswurzeln. Da wir auch Gleichungen höherer Ordnung wie zum Beispiel z^5-1, untersuchen wollen, ist es wichtig zu wissen, wie sich diese Einheitswurzeln berechnen. Allgemein wird eine n-te Wurzel (also n-Lösungen) aus einer komplexen Zahl $z=r*\exp(ß) = Re(z)+i*Im(z)$ folgendermaßen berechnet:

$$\sqrt[n]{z_m} = \sqrt[n]{r} * \exp\left(m*\frac{ß}{n}\right)$$

(mit m=Ø,1,2,3,..n)

Im Falle der Einheitswurzeln gilt r=1 und ß=36Ø°. Damit erhalten wir:

$$\sqrt[n]{z_m} = \exp\left(m*\frac{36Ø°}{n}\right) =$$
$$\cos\left(m*\frac{36Ø°}{n}\right) + i* \sin\left(m*\frac{36Ø°}{n}\right)$$



4-3-2

Wählen wir zum Beispiel n=3, so erhalten wir das bereits oben angegebene Ergebnis:

$z_1=1$, $z_2=.5*(-1+\sqrt{3}*i)$,
$z_3=.5*(-1-\sqrt{3}*i)$

Wie jetzt verständlich, liegen die Einheitswurzeln auf einem Kreis mit Radius 1 um den Ursprung (denn ihr Betrag ist immer 1) und bilden ein regelmäßiges n-Eck.

Zum Experimentieren hier noch die Einheitswurzeln bis n=5:

```
n
11
21-1
31(-.5+.866i) (-.5-.866i)
41i-1 -i
51   (.3Ø9+.951i)(-.8Ø9+.588i)(-.8Ø9-.588i)(.3Ø9-.951i)
(zu n=5, siehe Abbildung 4-3-2)
```

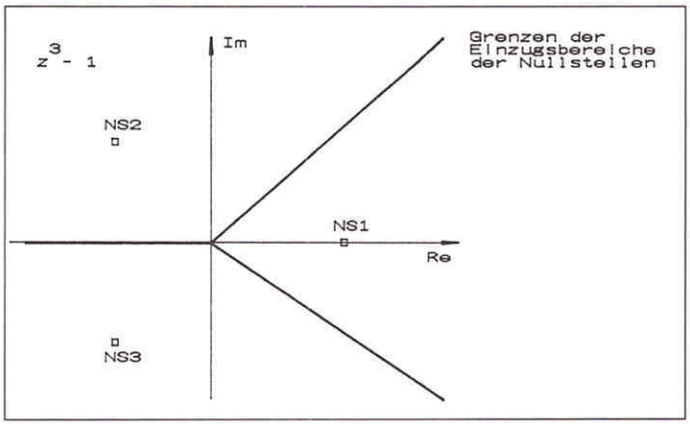
Programm: *wurz.c*

```
 1  /* Programm zur Berechnung der Einheitswurzeln */
 2  /* wurz.c                                      */
 3
 4  #include <math.h>
 5
 6
 7  main()
 8  {
 9   short n=6,i,j;
1Ø   float w,re,im;
11
12   printf("Berechnung der Einheitswurzeln von z=1 bis z%d=1\n",n);
13   for(i=1;i<=n;++i)
14   {
15      printf("\nz%-Ød=1\n",i);
16      for(j=Ø;j<i;++j)
17      {
18         w=2*j*PI/i;
19         re=cos(w),im=sin(w);
2Ø         printf(" z(%d)= %-1f + i*%-1f\n",j+1,re,im);
21      }
22   }
23  }
```

So, die Theorie ist erklärt, nun wird es wieder Zeit für ein wenig Praxis, also ein Programm.

Wir wollen wie gesagt das Verhalten der komplexen Nullstellensuche für die Funktion z^3-1 erforschen. Wir erwarten gerade in der Übergangszone zwischen den Einzugsbereichen der einzelnen Nullstellen ein interessantes Verhalten (siehe Abbildung 4-3-3). Um die komplexe Ebene ohne erneutes Kompilieren bequem untersuchen zu können, habe ich bei dem Programm Eingabemöglichkeiten der wichtigsten Parameter vorgesehen.

Außerdem kann wahlweise das Verhalten gegenüber nur einem Attraktor untersucht werden. Die Farbe, die ein Punkt erhält, wird



4-3-3

bestimmt durch die Anzahl der Iterationen die die Folge für diesen Startwert gebraucht hat, bis ein Abbruchkriterium erfüllt war. Waren zum Beispiel n=95 Durchgänge nötig und hat das Bild 32 mögliche Farben, so erhält dieser Punkt die Farbe des

(95)modulo (32-1) + 1 = 3.

Farbregisters. Die Subtraktion von 1 und die nachträgliche Addition ist sinnvoller als die Wahl des (95)modulo(32) = 32. Farbregisters, weil so verhindert wird, daß die Hintergrundfarbe (Register Ø) gewählt wird. Wen die ansonsten entstehenden »Löcher« nicht stören, muß dies natürlich nicht beachten.

Damit der Rechenaufwand für ein Bild in einem vernünftigem Rahmen bleibt, arbeiten wir zu Beginn nur mit der niedrigsten Auflösung. Die Verwendung einer anderen Auflösung erfolgt dann durch Setzen der entsprechenden Werte und Flags in den Zeilen 23 bis 26.

Soll das Bild nach dem Berechnen gespeichert werden, so ist die externe Funktion save() einzubinden und das Programm mit dem Programm *iff.c* zusammenzulinken.

Der Aufruf des Programms erfolgt zum Beispiel folgendermaßen:

```
newton1.run 1ØØ,.ØØ1,1ØØØ,-1,1,-1,1 1
```

Jetzt wird die Folge im Bereich Re(-1,1) und Im(-1,1) gegenüber der ersten Nullstelle (z=1) untersucht. Die Abbruchkriterien sind n_{max}=1ØØ, e_{min}=.ØØ1 und e_{max}=1ØØØ.

An dieser Stelle noch eine Anmerkung: Vergleicht man Bilder aus verschiedenen Büchern miteinander, so stellt man eventuell fest, daß die Orientierung verschieden ist. Dies liegt daran, daß nicht immer der Realteil nach links und der Imaginärteil nach oben gerichtet ist, oder Formeln mit anderem Vorzeichen (zum Beispiel z^2+c statt z^2-c) verwendet wurden.

Programm: *newton1.c*

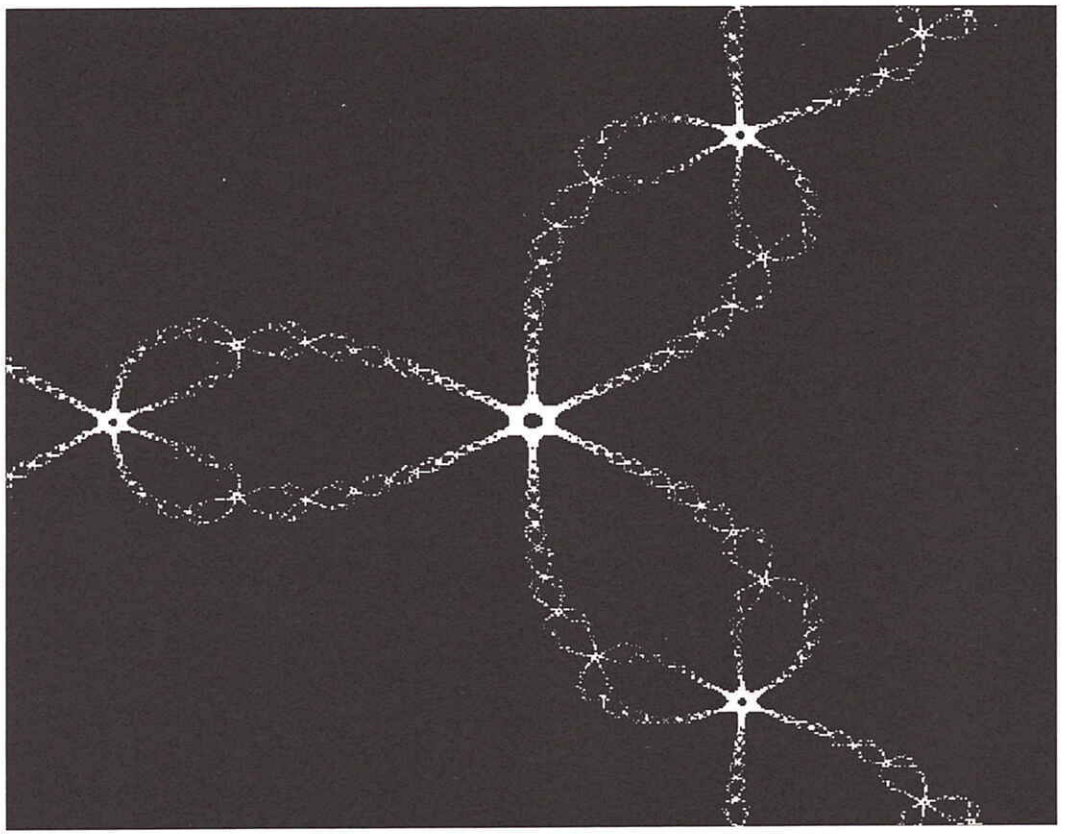
```
 1 /************************************************************************/
 2 /* newton1.c                                              Ø8.12.87   */
 3 /* Version 1.Ø                                                        */
 4 /*                                                                    */
 5 /*                                                                    */
 6 /* Auswertung einer komplexen Nullstellen-Suche an Hand der Funktion */
 7 /* z3+-c=Ø.  Untersuchung eines Attraktors mit entsprechender Farb-  */
 8 /* interpretation. (Farbe wird nach Anzahl der notwendigen Durch-     */
 9 /* laeufen bis zum Erreichen eines Attraktors errechnet.)             */
1Ø /*          Attraktor z=1                                             */
11 /*          Attraktor z=-.5+i*.866                                    */
12 /*          Attraktor z=-.5-i*.866                                    */
13 /*                                                                    */
14 /* fuer Markt+Technik geschrieben von Heiko Knappe                    */
15 /*                                                                    */
16 /* Komplexe Dynamische Systeme                                        */
17 /************************************************************************/
18
19
2Ø #include <exec/types.h>
21 #include <intuition/intuition.h>
22 #include <math.h>
```

```
23
24 #define WIDTH  32Ø          /* Screen Modus */
25 #define HEIGHT 25Ø
26 #define DEPTH  5
27 #define FLAGS  Ø            /* HIRES+LACE */
28
29 #define MIN Ø.ØØØØØØØ1      /* kritisch? */
3Ø
31 /* extern short save(); */
32
33 struct IntuitionBase *IntuitionBase;
34 struct GfxBase *GfxBase;
35 struct Screen *MyScreen;
36 struct RastPort *rp;
37 struct NewScreen MyNewScreen=
38  {Ø,Ø,WIDTH,HEIGHT,DEPTH,1,Ø,FLAGS,CUSTOMSCREEN,Ø,Ø,Ø,Ø};
39
4Ø int n,nmax=1ØØ;            /* Iterationsschritte                */
41 float emin=.Ø1,emax=1ØØØ;  /* Abbruchkriterium                  */
42 float rmin=-1,rmax=1;       /* Grenzen des Bildes                */
43 float imin=-1,imax=1;
44 float rst,ist;
45 short anzcol;               /* Anzahl der Farben                 */
46 short x=Ø,y=Ø;              /* x,y->Punktkoordinaten             */
47 float re,im,re1,re2,im1,im2,h1,h2;
48                            /* re,im laufender Re- und Im-Teil   */
49                            /* re1,re2,im1,im2  -> Folgenglieder */
5Ø                            /* h1,..h2 -> Hilfsvariable          */
51 float d[3];                /* Abstaende zu den Nullstellen      */
52 int   i,mode=Ø;            /* Bewertungsmodus                   */
53                            /* Ø -> Unendlich und alle Nullstellen */
54                            /* (1..3) -> Abstand zur i. NS       */
55
56 /************************/
57 /* Hauptprogramm       */
58 /************************/
59
6Ø main(argc,argv)
61 int argc;
62 char *argv[];
63 {
64   if((argc!=2)&&(argc!=1)&&(argc!=3))
65   {
66     printf("Eingabe: %s <nmax,emin,emax,rmin,rmax,imin,imax> <mode>\n",
67            argv[Ø]);
68     printf(" nmax: Abbruch nach welcher Anzahl von Schritten?\n");
69     printf(" emin: Attraktor?\n");
7Ø     printf(" emax: Unendlich?\n");
71     printf(" [rmin,rmax] [imin,imax]: Wertebereich der Bildebene\n");
72     printf(" mode: Ø-> zu allen NS, Unendlich\n
               i (1..3)-> zur i. NS\n");
73     close_things();
74   }
75   if(argc==2argc==3)
76     if(sscanf(argv[1],"%d,%f,%f,%f,%f,%f,%f",
77        &nmax,&emin,&emax,&rmin,&rmax,&imin,&imax)!=7)
78     {
79        printf("falsche Eingabe\n");
8Ø        close_things();
81     }
82   if(argc==3)
83     if((mode=sscanf(argv[2],"%d",&mode))!=1)
84        mode=Ø; /* ein unsinniger Wert bedeutet Ø */
85   if(mode>3mode<Ø)
86     mode=Ø;
87   printf("Parameter:\n nmax=%d\n emin=%f emax=%f\n",nmax,emin,emax);
88   printf(" [%f,%f] [%f,%f]\n",rmin,rmax,imin,imax);
89   printf(" mode: %d\n",mode);
9Ø   emin*=emin;  /* mit dem Quadrat zu vergleichen erspart Wurzeln zu */
91   emax*=emax;  /* berechnen -> Schnelligkeit                        */
92   open_things();
93   newton();
94   while(MyScreen->TopEdge<1ØØ);
95   /* falls das Bild gespeichert werden soll      */
96   /* save("Newton.pic",1,MyScreen);              */
97   close_things();
98 }
99
1ØØ
1Ø1 /************************/
1Ø2 /* Initialisierung     */
1Ø3 /************************/
```

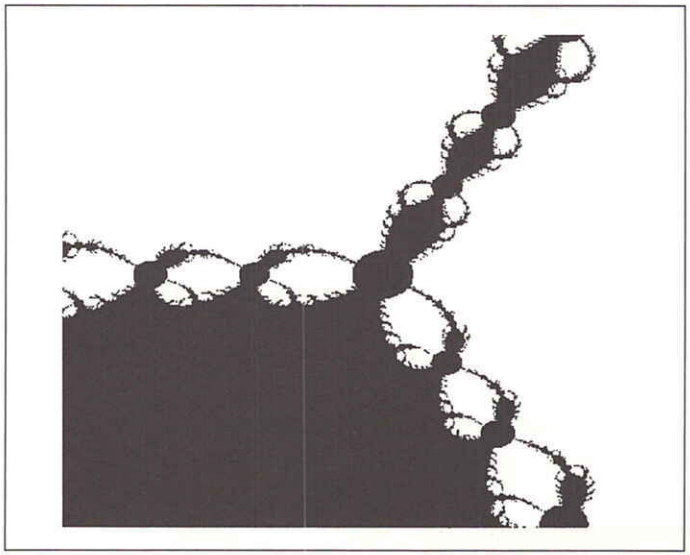
```
104
105 open_things()
106 {
107  if((IntuitionBase=OpenLibrary("intuition.library",0))==0)
108     close_things();
109  if((GfxBase=OpenLibrary("graphics.library",0))==0)
110     close_things();
111  if((MyScreen=OpenScreen(&MyNewScreen))==0)
112     close_things();
113  rp=&(MyScreen->RastPort);
114  init_colors();
115             (int)((1-(float)i/(float)anzcol)*16));
116
117
118 close_things()
119 {
120  if(IntuitionBase!=0) CloseLibrary(IntuitionBase);
121  if(GfxBase!=0) CloseLibrary(GfxBase);
122  if(MyScreen!=0) CloseScreen(MyScreen);
123  exit();
124 }
125
126
127 init_colors()
128 {
129  BYTE i;
130
131  for(i=0,anzcol=1;i<DEPTH;anzcol*=2,++i);
132  for(i=0;i<anzcol/2;++i)
133     SetRGB4(&(MyScreen->ViewPort),i,0,(int)((i*15)/(float)anzcol),0);
134  for(i=anzcol/2;i<anzcol;++i)
135     SetRGB4(&(MyScreen->ViewPort),i,5,(int)((i*15)/(float)anzcol),0);
136 }
137
138
139 /***********************/
140 /* Berechnung          */
141 /***********************/
142
143 newton()
144 {
145  float nsre[3],nsim[3];
146
147  nsre[0]=1,nsim[0]=0;              /* 1.Nullstelle */
148  nsre[1]=-.5,nsim[1]=.866;         /* 2.Nullstelle */
149  nsre[2]=-.5,nsim[2]=-.866;        /* 3.Nullstelle */
150  rst=(rmax-rmin)/WIDTH;
151  ist=(imax-imin)/HEIGHT;
152  for(x=0;x<WIDTH;++x)
153  {
154     re=rmin+x*rst;
155     for(y=0;y<HEIGHT;++y)
156     {
157        if(MyScreen->TopEdge>100)
158           return(1);
159        im=imin+y*ist;
160        SetAPen(rp,anzcol-1);
161        WritePixel(rp,x,y);
162        for(n=0,re1=re,im1=im;n<nmax;++n)
163        {
164           /* Berechnung des naechsten Folgeglieds */
165           h1=re1*re1+im1*im1;
166           if(h1>MIN)
167           {
168              re2=(2*re1+(re1*re1-im1*im1)/(h1*h1))/3;
169              im2=(2*im1-(2*re1*im1)/(h1*h1))/3;
170              re1=re2;
171              im1=im2;
172           }
173           else                  /* eigentliche Nullstelle?   */
174              break;
175           /* Abstaende des neuen Folgenglieds zu den einzelnen */
176           /* Attraktoren, das heisst Nullstellen.              */
177           for(i=0;i<3;++i)
178           {
179              h1=re1-nsre[i];       /* i. Nullstelle          */
180              h2=im1-nsim[i];
181              d[i]=h1*h1+h2*h2;
182           }
183           /* Ist ein Attraktor erreicht?                       */
184           if(d[0]<emind[1]<emind[2]<emin)
185              break;
186           /* Attraktor Unendlich?                              */
```

```
187             if(d[Ø]>emaxd[1]>emaxd[2]>emax)
188                 break;
189         }
19Ø         SetAPen(rp,Ø);
191         if((mode==Ø)(d[mode]<emin))
192             SetAPen(rp,n%(anzcol-1)+1);
193         WritePixel(rp,x,y);
194     }
195 }
196 }
```

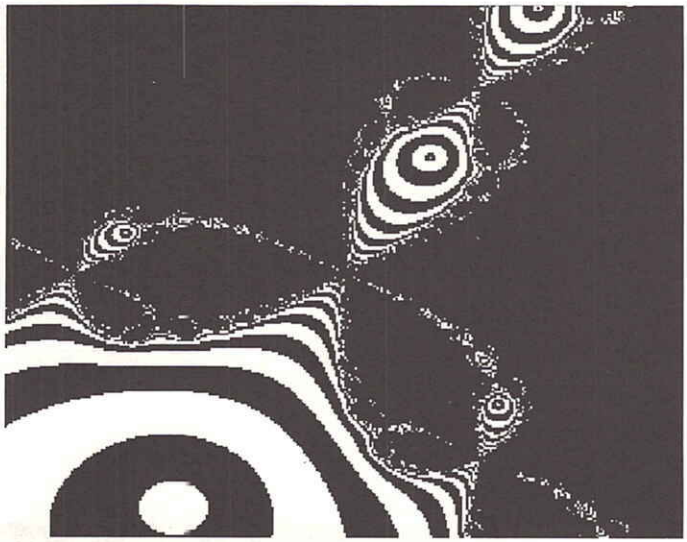
Starten Sie nun doch einmal das Programm und beobachten sie, wie das Bild entsteht – es ist faszinierend, oder? Betrachten wir das fertige Bild, erkennen wir die drei Nullstellen (oder auch nur eine je nach gewähltem Modus) und die Grenzbereiche zwischen den Einzugsgebieten. Wie zu erwarten, weisen die Grenzen die interessantesten Strukturen auf. Auch wenn wir die Grenze vergrößern, also gleichsam in die komplexe Ebene zoomen, erhalten wir immer wieder Figuren, die sich zwar unterscheiden, aber doch Ähnlichkeiten aufweisen. Wieder haben wir einen Beweis für die Selbstähnlichkeit und die Ordnung im



4-3-4 Fraktale Grenze ($z^3-1=0$)



4-3-5 Einzugsbereich einer Nullstelle (-0,5+0,866i)



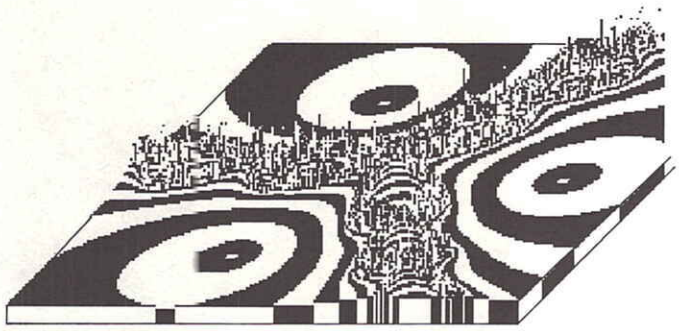
4-3-6 (1) Höhenlinien/Geschwindigkeitsdiagramm

Chaos der dynamischen Systeme entdeckt. Und noch etwas Wichtiges: Interessant sind wirklich nur die Grenzen. Sie sind bei genauer Betrachtung so zerstückelt und zerklüftet als ob in ihrem Bereich nie nur zwei Attrationsgebiete aneinanderliegen könnten, sondern auch das dritte immer angrenzen müßte. Es ist schon seltsam, daß die Folge zu dem am weitesten entfernten Attraktor läuft.

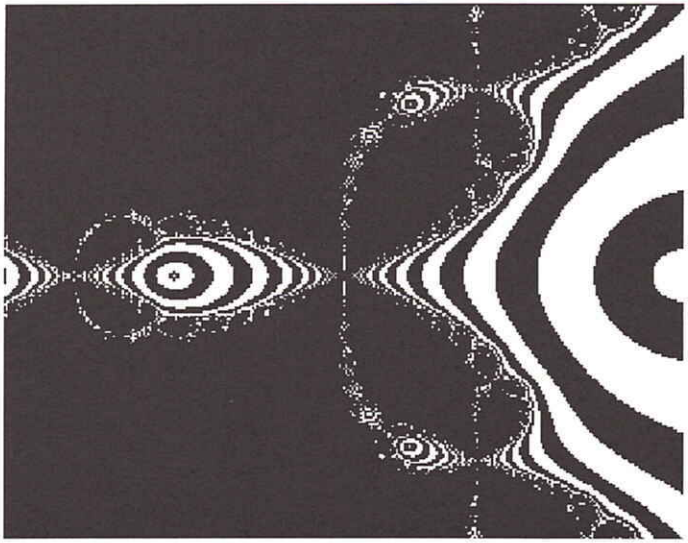
Hier wird offensichtlich, wie unendlich viele verschiedene Grafiken und Interpretationen möglich sind. Interessant ist es zum Beispiel, nicht die Konvergenz gegen alle Nullstellen, sondern nur gegen eine zu untersuchen, oder die Einzugsbereiche mit drei verschiedenen Farben (z.B. Rot, Grün, Blau) zu kennzeichnen, oder nur die Punkte zu zeichnen für die die Berechnung auf Grund der Überschreitung von n_{max} abgebrochen wurde. Die letzte Interpretation liefert uns dann zum Beispiel nur den Grenzverlauf (siehe Bild 4-3-4).

Mehr gibt es eigentlich zu der Nullstellenuntersuchung der Funktion z^3-1 nicht zu sagen, lassen sie sich doch von den Abbildungen 4-3-5 bis 4-3-10 verzaubern!

Sobald wir unsere Entdeckungslust an dieser Funktion ausgelassen haben, ist unserem Forschungsdrang immer noch keine Grenze gesetzt. Jetzt geht's erst los! Warum nicht



4-3-7 (1) 3D-Bild ($z^3-1=0$) Nullstellen und Grenzen sind deutlich zu erkennen.

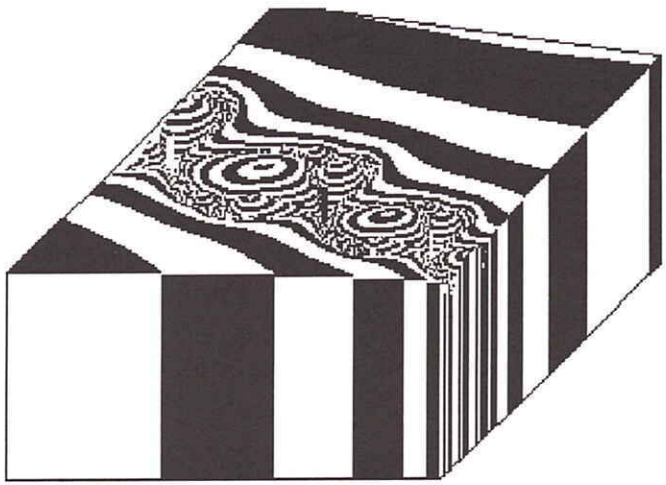


4-3-6 (2) Nullstelle (reelle Nullstelle ,1)

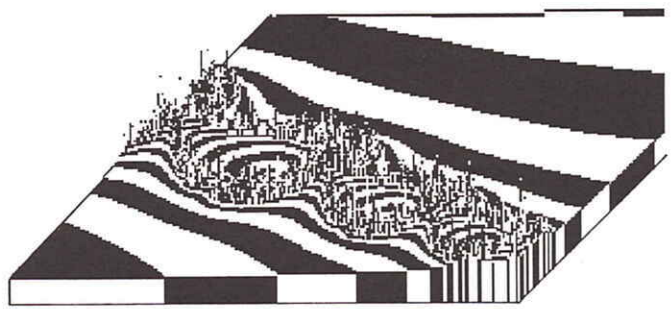
4-3-6 (3) Nullstelle (-0,5-0,866i)

Funktionen höheren Grades wie $z^4-1=\emptyset$ oder $z^5-1=\emptyset$ untersuchen (siehe Abbildungen 4-3-13 bis 4-3-15)? Hier erhalten wir natürlich wieder sehr harmonische und symmetrische Grenzverläufe. Aber warum sollten wir nicht einmal eine Funktion $z^3-3*z^2-z-1=\emptyset$ untersuchen? Oder gar die Formel des Newton-Verfahrens ändern, zum Beispiel einen komplexen Faktor c einfügen:

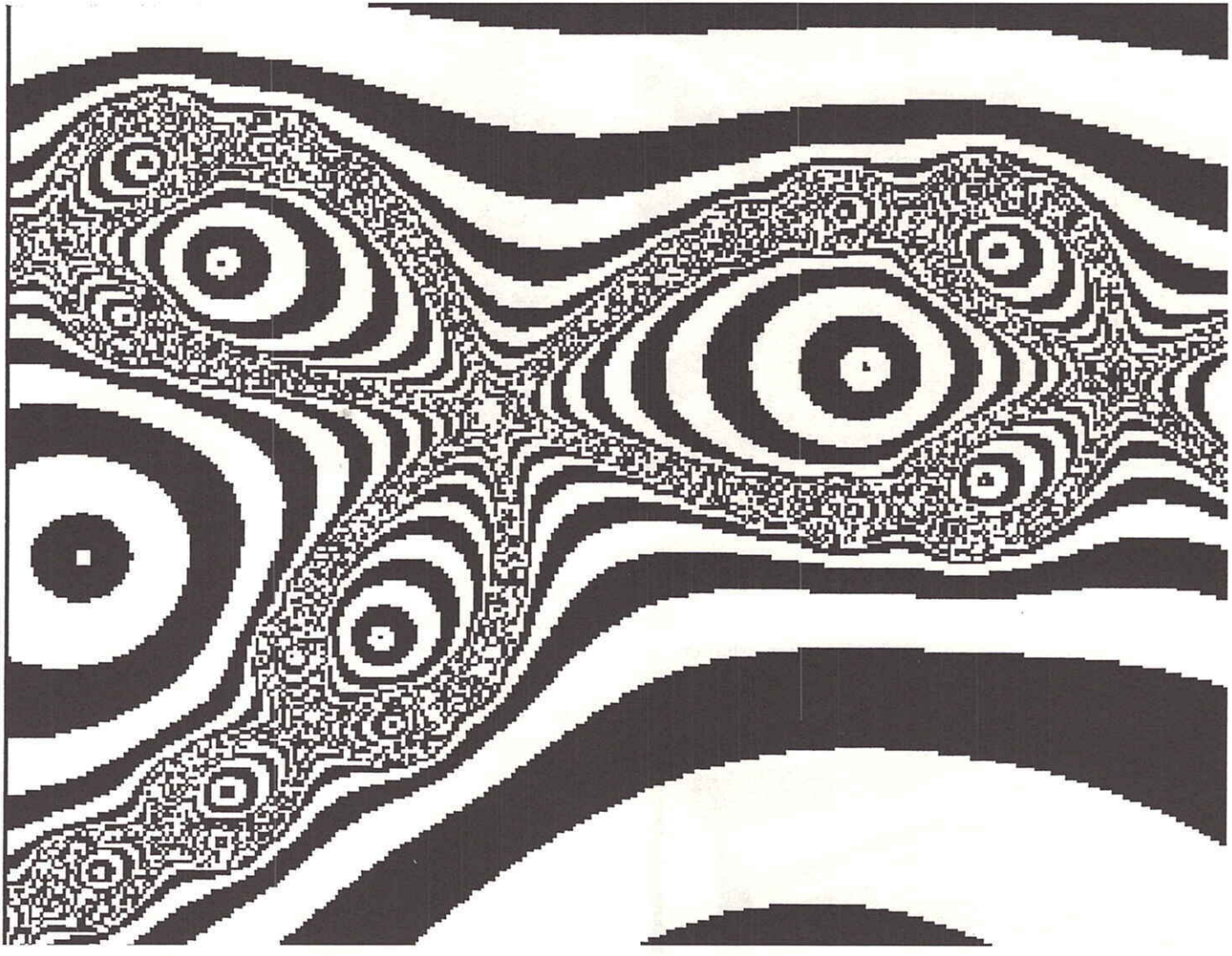
$$x_{n+1} = x_n - \frac{f(x_n)}{(c*f'(x_n))}$$



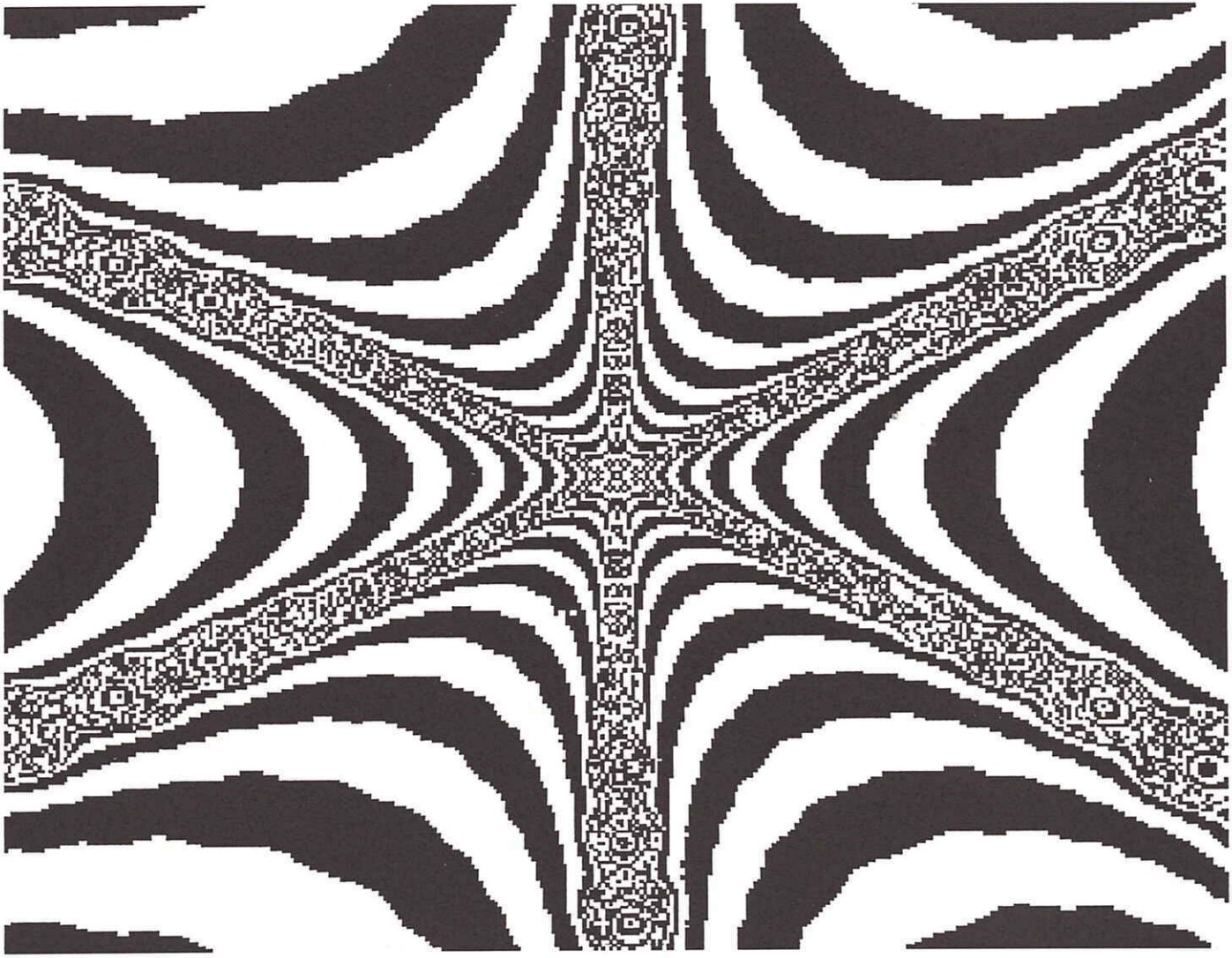
4-3-7 (2) 3-D-Interpretation eines Grenzausschnitts (negative Darstellung)



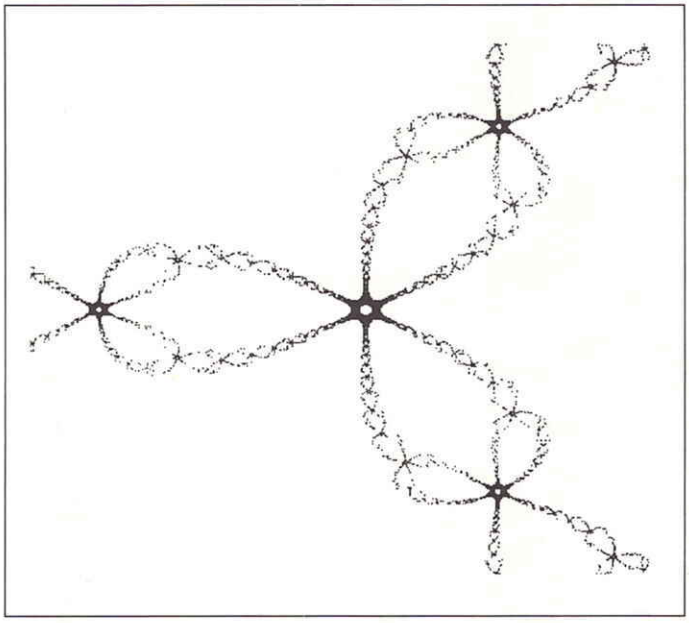
4-3-7 (3) Positive Darstellung zu Bild 4-3-7 (1)



4-3-8 Ebene Ausschnittvergößerung (z^3-1=0)
Deutlich zu erkennen ist die Selbstähnlichkeit der Grenzstruktur



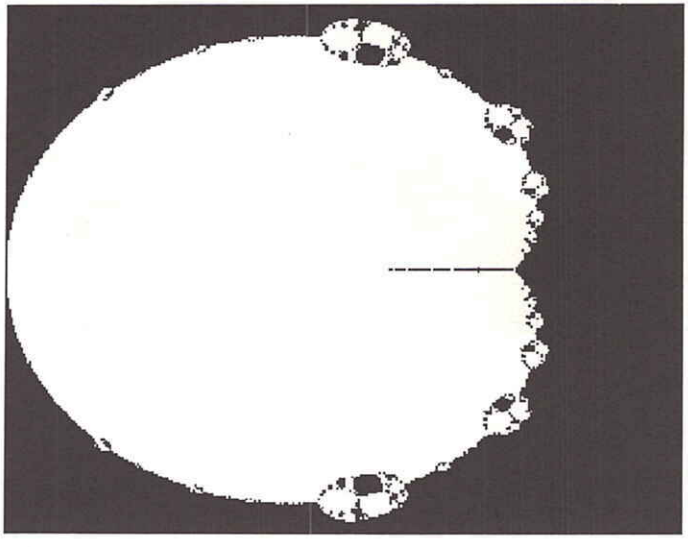
4-3-9 Ebene Ausschnittsvergrößerung aus dem Zentrum (z^3-1=0)



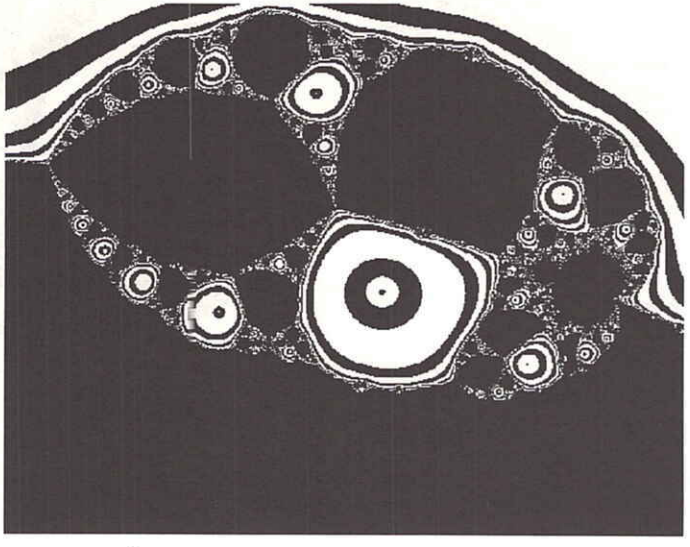
4-3-10 Die Grenzen der Attraktionsgebiete

Jeden, den diese Bilder faszinieren, der die innere Ordnung, die Selbstähnlichkeit und die unendliche Vielfalt liebt, kann mit ein wenig Kreativität immer neue Entdeckungen machen und sollte sich durch die notwendige Vorarbeit nicht abschrecken lassen. Sicher ist hier die Grenze zwischen Forschung, Theorie und Wissenschaft auf der einen Seite und Kreativität und Kunst auf der anderen erreicht!

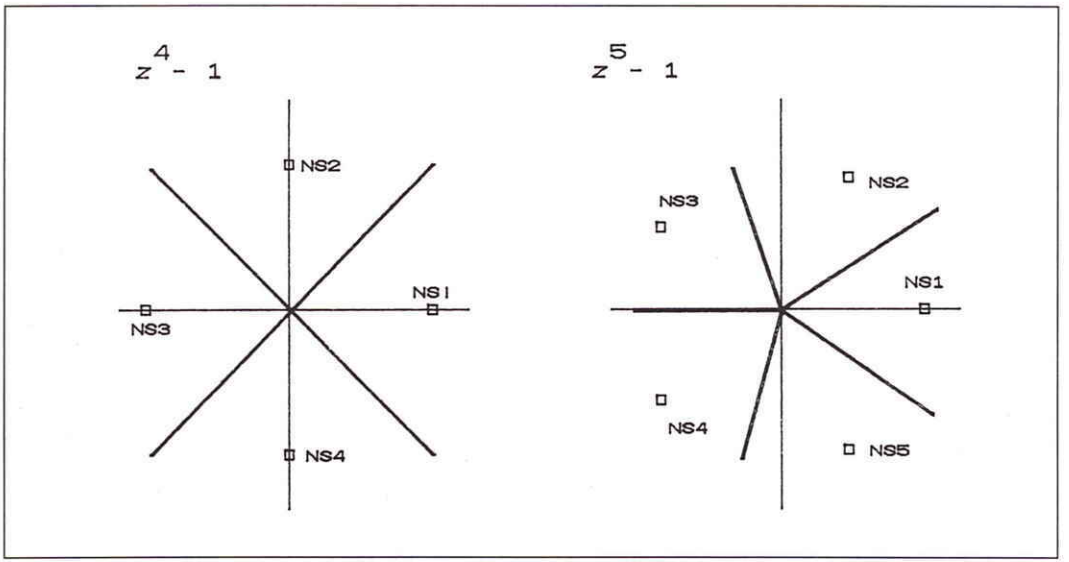
Zum Abschluß noch einige Ideen und Formeln, die ebenfalls die Eigenschaft der Rückkopplungen zeigen und ähnliche Ergebnisse wie die Auswertung des Newton-Verfahrens aufweisen. Die Rechenzeit steigt bei ihnen recht schnell an, so daß man durchaus Zeiten bis zu 10 Stunden erwarten kann, auch wenn man keine extremen Vergrößerungen vornimmt.



4-3-11 (1) Sullivan – Untersuchung der Konvergenz (Divergenz) gegen 1 ($z^3+(c-1)\,z-c=1$)



4-3-11 (2) Überraschung: rechts deutlich ein Apfelmännchen zu erkennen (Vergrößerung der großen oberen Knospe aus 4-3-11 (1))



4-3-12 Grenzverlauf und Nullstellen für $z^4-1=0$ und $z^5-1=0$

Als erstes eine Auswertung einer etwas umfangreicheren Formel mit dem Newton-Verfahren. Nach einer Arbeit von Curry, Sullivan und Garnett können alle Gleichungen dritten Grades mit der Formel

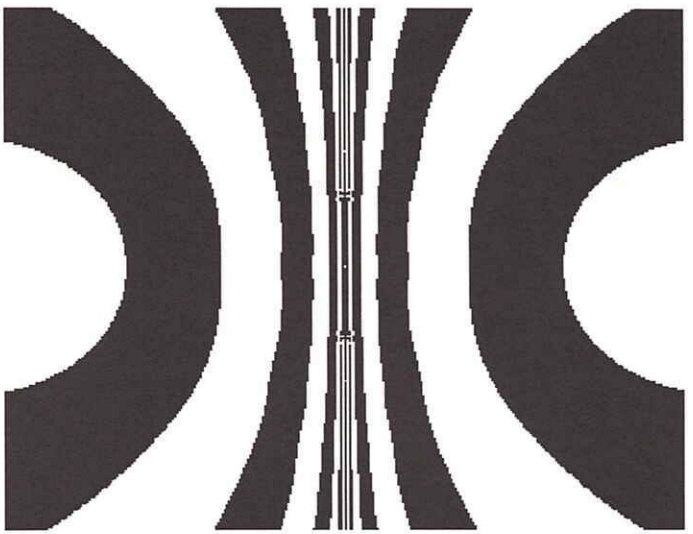
$f(z)=z^3+(c-1)*z-c$

dargestellt werden. Die Variable »c« in der Funktion ist wie »z« ein komplexer Wert. Wir können nun diese Formel mit Hilfe des Newton-Verfahrens untersuchen. Dabei ergibt sich folgende Gleichung:

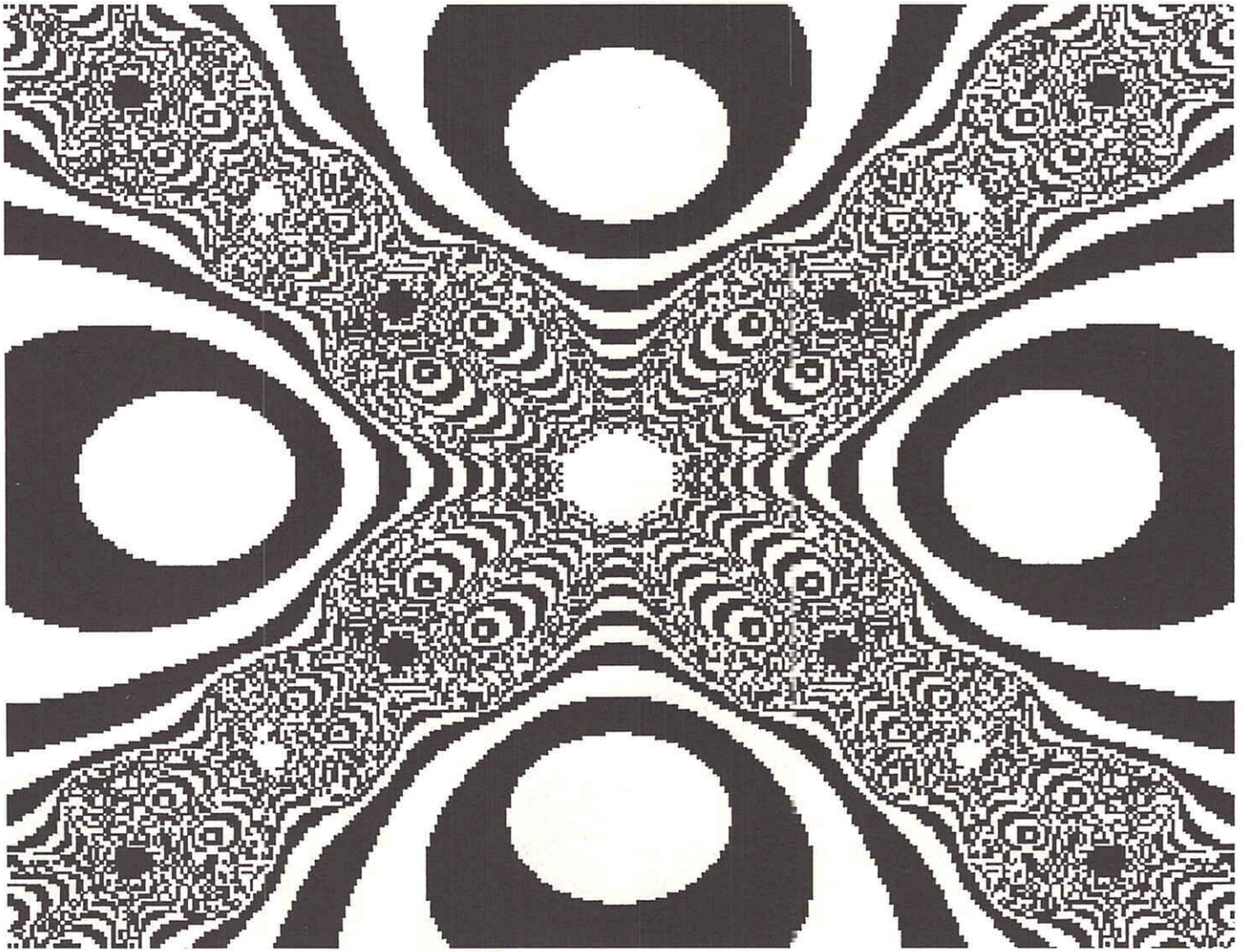
$$z_{n+1} = z_n - \frac{f(z_n)}{f'(z_n)} = z_n - \frac{z_n^3+(c-1)*z_n-c}{3*z_n^2+(c-1)}$$

Jetzt erkennt man schon den Mehraufwand, den diese Iteration erfordert, denn die reelle Darstellung des Nenners zu erhalten macht einiges Rechnen notwendig.

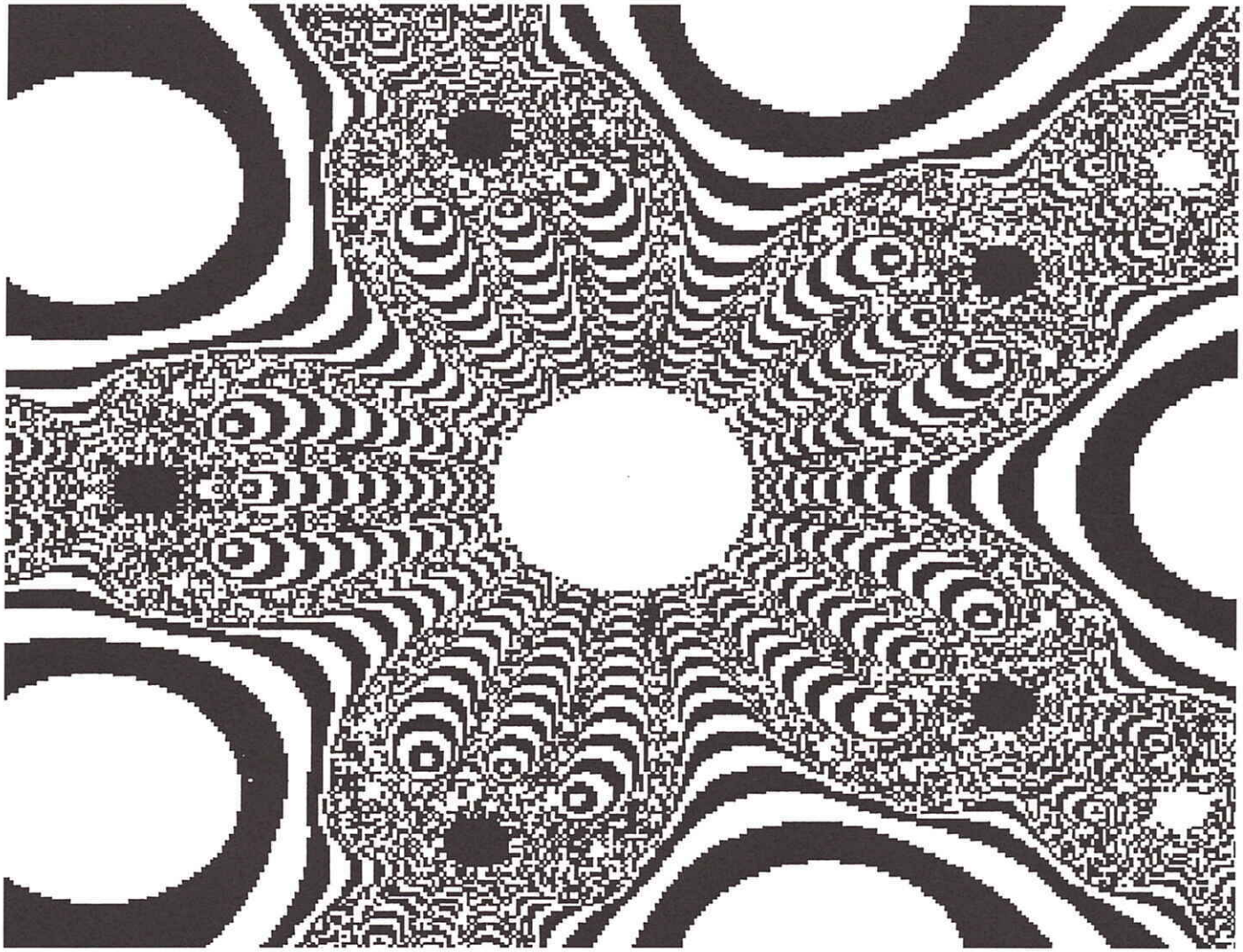
Als Hilfswerte werden die Variablen `ReN`, `ImN` (Real- und Imaginärteil des Nenners) und



4-3-13 $z^2-1=0$



4-3-14 $z^4-1=0$



4-3-15 $z^5-1=0$

ReZ, ImZ (Real- und Imaginärteil des Zählers) eingeführt.

Als Voraussetzung geben wir uns z=Re+Im*i und c=c_{re} + c_{im}*i vor. Damit erhalten wir:

$$ReN = 3*(Re^2-Im^2) + (c_{re}-1)$$
$$ImN = 6*Re*Im + c_{im}$$
$$ReZ = Re^3 - 3*Re*Im^2 + (c_{re}-1)*Re - c_{im}*Im - c_{re}$$
$$ImZ = 3*Re^2*Im - Im^3 + (c_{re}-1)*Im + c_{im}*Re - c_{im}$$

und das neue Folgenglied

$$Re(z_{n+1}) = Re - \frac{(ReZ*ReN + ImZ*ImN)}{(ReN^2 + ImN^2)}$$

$$Im(z_{n+1}) = Im - \frac{(ImZ*ReN - ReZ*ImN)}{(ReN^2 + ImN^2)}$$

Da wir jetzt eine Folge vor uns haben, die von zwei Parametern (»c« und »z«) abhängt, sind wir in der Lage sie auf zwei verschiedene Arten zu interpretieren. Erstens können wir die Nullstellen von f(z) untersuchen, indem wir die Zeichenebene als z-Ebene wählen, also c als konstant setzen und das erste Folgenglied durch den Bildpunkt bestimmen.

Die zweite Möglichkeit besteht darin, die Bildebene als komplexe c-Ebene zu wählen, also das erste Folgenglied konstant zu setzen. Auch die Formel z^2-c bietet diese Möglichkeiten. Wie wir noch sehen werden, führt die erste Interpretation zu den Apfelmännchen, die zweite zu Julia-Mengen.

Untersuchen wir nun das Verhalten unserer obigen Formel in der c-Ebene, also nach dem zweiten Verfahren, so erwarten wir vor allem für betragsmäßig große c-Werte Konvergenz gegen die Nullstelle 1. Außer diesem Attraktor existieren noch zwei weitere, gegen die die Folge konvergieren kann. Des weiteren ist es möglich, daß die Folgenglieder periodisch immer wieder auftreten (das heißt $z_{n+m}=z_n$).

Das folgende Programm wertet die Konvergenz gegen den Attraktor 1 auf bekannte Weise aus.

Programm: *sullivan.c*

```
/*********************************************************************/
/* sullivan.c                                          Ø8.Ø1.88  */
/* Version 1.Ø                                                   */
/*                                                               */
/*                                                               */
/* Auswertung einer komplexen Nullstellen-Suche an Hand der Funktion */
/* z3+(c-1)-c=Ø.                                                 */
/* (fuer grosse c ist z=1 Attraktor)                             */
/*                                                               */
/*                                                               */
/* fuer Markt+Technik geschrieben von Heiko Knappe               */
/*                                                               */
/* Komplexe Dynamische Systeme                                   */
/*********************************************************************/


#include <exec/types.h>
#include <intuition/intuition.h>
#include <math.h>


#define WIDTH  64Ø          /* Screen Modus */
#define HEIGHT 4ØØ
#define DEPTH  3
#define FLAGS  HIRES+LACE

#define MIN Ø.ØØØØØØØ1      /* kritisch? */

extern short save();

struct IntuitionBase *IntuitionBase;
struct GfxBase *GfxBase;
struct Screen *MyScreen;
struct RastPort *rp;
struct NewScreen MyNewScreen=
 {Ø,Ø,WIDTH,HEIGHT,DEPTH,1,Ø,FLAGS,CUSTOMSCREEN,Ø,Ø,Ø,Ø};
```

```
37
38 int i,n,nmax=5Ø;          /* Iterationsschritte      */
39 float emin=.ØØØ1;         /* Abbruchkriterium        */
4Ø float rmin=-2,rmax=2;     /* Grenzen des Bildes      */
41 float imin=-2,imax=2;
42 float rst,ist;
43 float nsre=1,nsim=Ø;      /* Attraktor               */
44 float d;                  /* Abstande zum Attraktor 1 */
45 short anzcol;             /* Anzahl der Farben       */
46 short x=Ø,y=Ø;            /* x,y->Punktkoordinaten   */
47 float re,im,rez,ren,imz,imn,h,h1,h2;
48 float cre,cim;
49
5Ø
51 /***********************/
52 /* Hauptprogramm       */
53 /***********************/
54
55 main(argc,argv)
56 int argc;
57 char *argv[];
58 {
59  if((argc!=2)&&(argc!=1))
6Ø  {
61     printf("Eingabe: %s <nmax,emin,rmin,rmax,imin,imax>\n",
62           argv[Ø]);
63     printf(" nmax: Abbruch nach welcher Anzahl von Schritten?\n");
64     printf(" emin: Attraktor?\n");
65     printf(" [rmin,rmax] [imin,imax]: Wertebereich der Bildebene\n");
66     close_things();
67  }
68  if(argc==2)
69     if(sscanf(argv[1],"%d,%f,%f,%f,%f,%f",
7Ø        &nmax,&emin,&rmin,&rmax,&imin,&imax)!=6)
71     {
72        printf("falsche Eingabe\n");
73        close_things();
74     }
75  printf("Parameter:\n nmax=%d\n emin=%f\n",nmax,emin);
76  printf(" [%f,%f] [%f,%f]\n",rmin,rmax,imin,imax);
77  emin*=emin;
78  open_things();
79  sullivan();
8Ø  while(MyScreen->TopEdge<1ØØ);
81  save("Sullivan.pic",1,MyScreen);
82  close_things();
83 }
84
85
86 /***********************/
87 /* Initialisierung     */
88 /***********************/
89
9Ø open_things()
91 {
92  if((IntuitionBase=OpenLibrary("intuition.library",Ø))==Ø)
93     close_things();
94  if((GfxBase=OpenLibrary("graphics.library",Ø))==Ø)
95     close_things();
96  if((MyScreen=OpenScreen(&MyNewScreen))==Ø)
97     close_things();
98  rp=&(MyScreen->RastPort);
99  init_colors();
1ØØ }
1Ø1
1Ø2 close_things()
1Ø3 {
1Ø4  if(IntuitionBase!=Ø) CloseLibrary(IntuitionBase);
1Ø5  if(GfxBase!=Ø) CloseLibrary(GfxBase);
1Ø6  if(MyScreen!=Ø) CloseScreen(MyScreen);
1Ø7  exit();
1Ø8 }
1Ø9
11Ø init_colors()
111 {
112  for(i=Ø,anzcol=1;i<DEPTH;anzcol*=2,++i);
113  for(i=Ø;i<anzcol/2;++i)
114     SetRGB4(&(MyScreen->ViewPort),i,(int)((i*15)/(float)anzcol),Ø,Ø);
115  for(i=anzcol/2;i<anzcol;++i)
116     SetRGB4(&(MyScreen->ViewPort),i,(int)((i*15)/(float)anzcol),Ø,8);
117 }
118
```

```
119
120 /**********************/
121 /* Berechnung         */
122 /**********************/
123
124 sullivan()
125 {
126  rst=(rmax-rmin)/WIDTH;
127  ist=(imax-imin)/HEIGHT;
128  for(x=0;x<WIDTH;++x)
129  {
130     cre=rmin+x*rst;
131     for(y=0;y<HEIGHT;++y)
132     {
133        if(MyScreen->TopEdge>100)
134           return(1);
135        cim=imin+y*ist;
136        SetAPen(rp,anzcol-1);
137        WritePixel(rp,x,HEIGHT-y-1);
138        for(n=re=im=0;n<nmax;++n)
139        {
140           /* Berechnung des naechsten Folgeglieds          */
141           rez=re*re*re-3*re*im*im+(cre-1)*re-cim*im-cre;
142           imz=3*re*re*im-im*im*im+(cre-1)*im+cim*re-cim;
143           ren=3*(re*re-im*im)+cre-1;
144           imn=6*re*im+cim;
145           h=ren*ren+imn*imn;
146           if(h>MIN) /* kritischer Punkt */
147           {
148              re=re-(rez*ren+imz*imn)/h;
149              im=im-(imz*ren-rez*imn)/h;
150           }
151           else                  /* eigentliche Nullstelle?  */
152              break;
153           h1=re-nsre;           /* Abstand zum Attraktor 1  */
154           h2=im-nsim;
155           d=h1*h1+h2*h2;
156           /* Ist ein Attraktor erreicht?                    */
157           if(d<emin)
158              break;
159        }
160        if(d<emin) SetAPen(rp,n%(anzcol-1)+1);
161        else SetAPen(rp,0);
162        WritePixel(rp,x,HEIGHT-y-1);
163     }
164  }
165 }
```

Bei der Vergrößerung der Grundfigur stellen wir etwas Erstaunliches fest. Der weiße Bereich konvergiert gegen eins. Wir erkennen links eine Figur, die wir im weiteren in einem etwas anderen Zusammenhang kennenlernen werden: ein Apfelmännchen. In ihr zeigt die Folge ein zyklisches Verhalten.

Für die anderen schwarzen Flächen konvergiert die Folge gegen eine der anderen Nullstellen.

Auch andere Formeln sind es wert, untersucht zu werden. Sie ergeben sich aus physikalischen Modellen für Magnetismus. Peitgen und Richter schlagen die Untersuchung der folgenden beiden Modellgleichungen vor:

Modell 1:

$$z_{n+1} = \left(\frac{z_n^2 + c - 1}{2 * z_n + c - 2} \right)^2$$

Modell 2:

$$z_{n+1} = \left(\frac{z_n^3 + 3*(c-1)*z_n + (c-1)*(c-2)}{3*z_n^2 + 3*(c-2)*z_n + c^2 - 3*c + 3} \right)^2$$

Auch hier hängen die Folgen von zwei komplexen Parametern ab, es ist also auch hier wieder möglich, ein und dieselbe Formel auf verschiedene Arten zu untersuchen.

Anmerkungen zum Newton-Verfahren

Sicher will jeder mit möglichst wenig Vorarbeit auch komplexe Formeln mit dem Newton-Verfahren auf ihr Verhalten bzgl. Nullstellen untersuchen. Deshalb möchte ich hier

als Beispiele noch die beiden Funktionen $z^4-1=\emptyset$ und $z^5-1=\emptyset$ ableiten.
(im folgenden gilt: z = x + i*y)

(1) $z^4-1=\emptyset$

Nach den Newton-Verfahren gilt:

$$z_{n+1} = z_n - \frac{(z_n^4-1)}{4*z_n^3} = \frac{1}{4} * \left(3*z_n + \frac{1}{z_n^3}\right)$$

Nun führen wird zwei Hilfsgrößen ein, um die Berechnung zu vereinfachen:

$$Re = x_n^3-y_n^2*x_n - 2*x_n*y_n^2$$
$$Im = x_n^2*y_n-y_n^3 + 2*x_n^2*y_n$$

Mit diesen beiden Größen erhalten wir für das neue Folgenglied:

$$z_{n+1} = x_{n+1} + i*y_{n+1}$$
$$x_{n+1} = \frac{1}{4} * \left(3*x_n + \frac{Re}{Re^2+Im^2}\right)$$
$$y_{n+1} = \frac{1}{4} * \left(3*y_n + \frac{Im}{Re^2+Im^2}\right)$$

(2) $z^5-1=\emptyset$

Nach den Newton-Verfahren gilt:

$$z_{n+1} = z_n - \frac{z_n^5-1}{5\ z_n^4} = \frac{1}{5} * \left(4*\ z_n + \frac{1}{z_n^4}\right)$$

Nun führen wird wieder zwei Hilfsgrößen ein, um die Berechnung zu vereinfachen:

$$Re = (x_n^2-y_n^2)^2 - 4*x_n^2*y_n^2$$
$$Im = 4*x_n*y_n*(x_n^2-y_n^2)$$

Mit diesen beiden Größen erhalten wir für das neue Folgenglied:

$$z_{n+1} = x_{n+1} + i*y_{n+1}$$
$$x_{n+1} = \frac{1}{5} * \left(4*x_n + \frac{Re}{Re^2 + Im^2}\right)$$
$$y_{n+1} = \frac{1}{5} * \left(4*y_n + \frac{Im}{Re^2 + Im^2}\right)$$

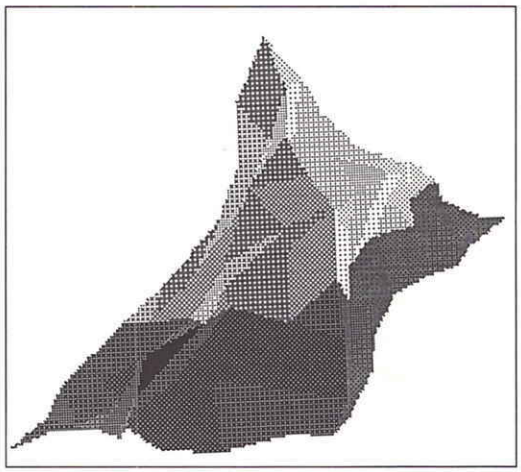
4.4 Ausblicke

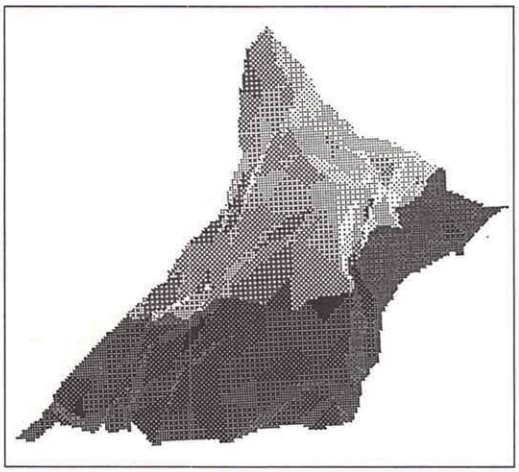
Wir sind nun am Ende unseres Vorstoßes ins Chaos der komplexen Systeme angekommen. Man könnte, wie wir inzwischen wissen, die Verfahren und Ideen zur Erzeugung »schöner« Bilder endlos fortsetzen. Sicher werden Sie selbst inzwischen eigene Erfahrungen gesammelt und vielleicht sogar neue Entdeckungen gemacht haben. Zum Abschluß möchte ich noch einige Systeme vorstellen, die ebenfalls die grundlegenden Eigenschaften fraktaler Grafik besitzen, jedoch mit rekursiven Folgen auf den ersten Blick nichts zu tun haben. Sie beruhen im Endeffekt natürlich wieder auf Rekursivität, denn das mehrfache Durchlaufen eines Algorithmus unter Zugrundelegen seines vorherigen Ergebnisses ist das gleiche Prinzip.

4.4.1 Landschaften aus dem Computer

In den letzten Jahren entwickelte sich aufgrund fortschreitender Fähigkeiten (vor allem Geschwindigkeit und Kapazität) der Mikroprozessoren ein neues Einsatzgebiet des Computers: die Computergrafik. Der Stand, der heute auf diesem Gebiet erreicht ist, wurde vor wenigen Jahren noch als Traum abgetan. Wie realistisch die Effekte sind, zeigt der Einsatz der Computergrafik im Film. Diese Wirklichkeitsnähe ist natürlich nur mit einem recht hohen Aufwand an Rechenzeit und Programmierung zu erreichen. Es ist also kaum möglich, den Profis auf diesem Gebiet so ohne weiteres Konkurrenz zu machen. Dennoch sollten wir uns dadurch nicht den Mut nehmen lassen. Auch mit einfachen Algorithmen ist es oft möglich, akzeptable Ergebnisse zu erzielen. Gerade der Amiga bietet als Grafiksystem die dazu notwendigen Möglichkeiten. Wir wollen uns aber nicht zu weit von unserem ei-

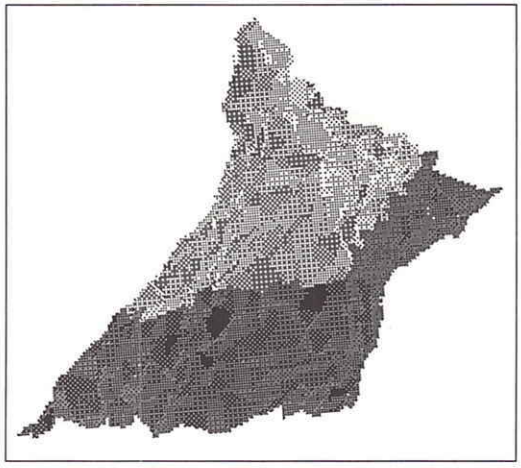
4-4-1-1 (1) Fraktale Landschaften (Grundelement sind Dreiecke)

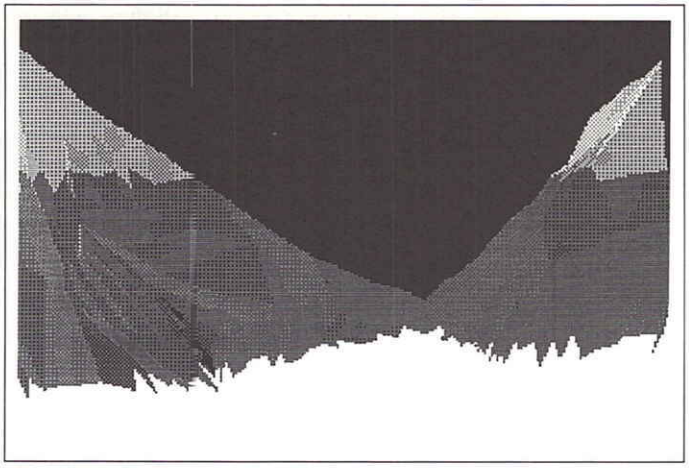




4-4-1-1 (2) feine Unterteilung . . .

4-4-1-1 (3) . . . und noch feiner



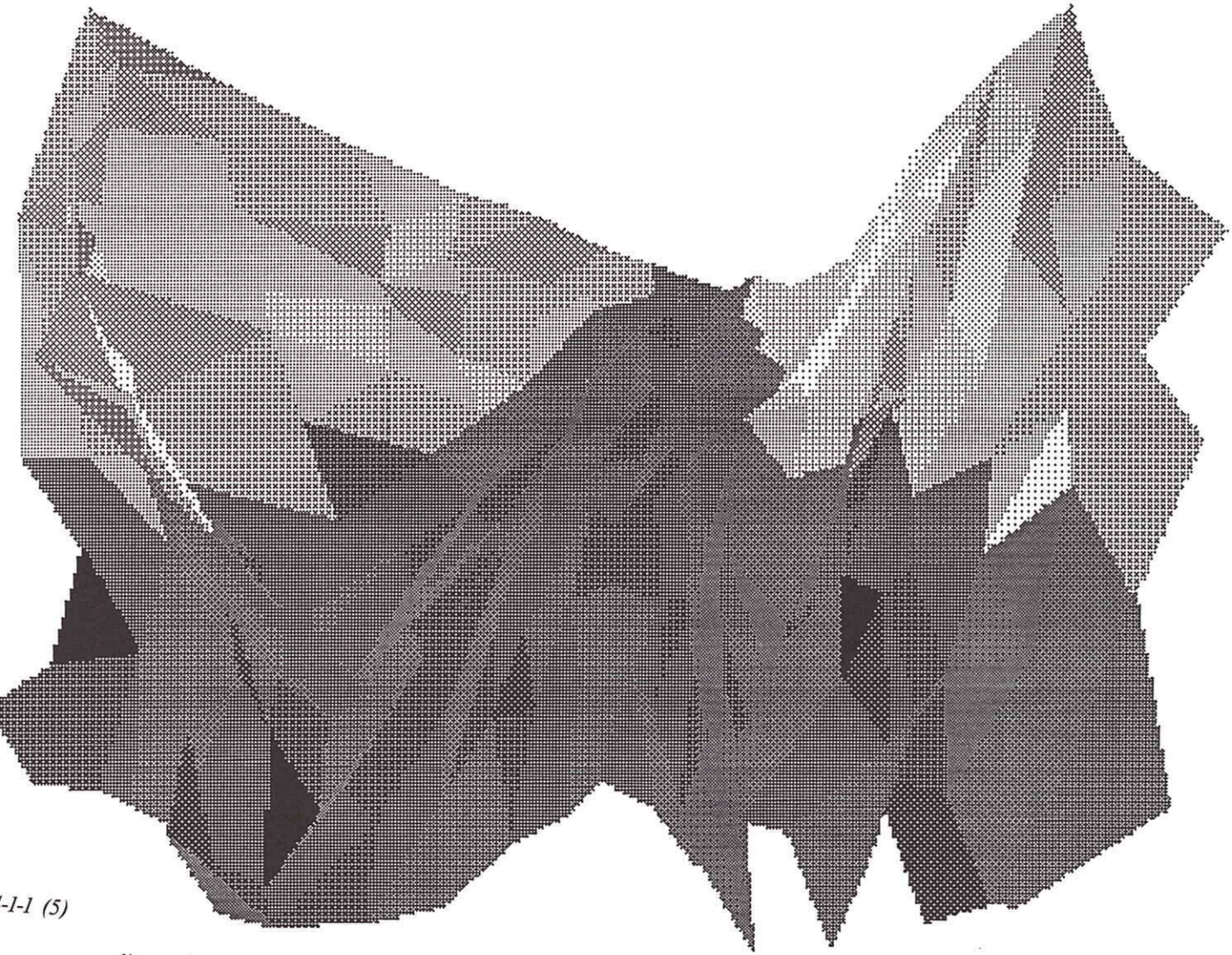


4-4-1-1 (4) Ein idyllischer Bergsee?

gentlichen Problem entfernen – der Untersuchung dynamischer Systeme. Stellen wir uns doch einmal folgende Aufgabe: Wir wollen versuchen, eine Landschaft künstlich und möglichst realistisch nachzubilden. Wie wir bereits festgestellt haben, ist ein wichtiges Element der Natur die innere Ordnung, ein ständiges Wiederholen und Verändern eines Grundbausteins. Deshalb wirken die Apfelmännchen ja so seltsam anziehend. Das entscheidende Element unseres Algorithmus wird also sein, durch Rückkoppeln Selbstähnlichkeit des Bildes zu erreichen. Beispielhaft möchte ich hier zwei Methoden vorstellen, eine Landschaft mit Hilfe des Computers zu schaffen:

1) Verwendung von Dreiecken

Wie oben schon angesprochen, wollen wir, ausgehend von einer Grundstruktur, unter Verwendung immer gleicher Bauelemente nach mehreren Iterationen eine Figur erhalten, die eine Landschaft nachahmt, denn mit



-1-1 (5)

dieser einfachen Methode ist sicher kein hundertprozentiges Modell zu erhalten. Wir verwenden als Grundstruktur ein Flächenmodell, welches aus Dreiecken zusammengesetzt ist. Die Punktetabelle und die Kantenzeiger werden am besten so abgelegt, wie bei der Einführung in die Grundlagen dreidimensionaler Grafik besprochen. Das Bauelement, welches wir hier verwenden, ist also ein Dreieck. In den nun folgenden Annäherungen müssen wir uns eine Arbeitsweise einfallen lassen, die es möglich macht, die Dreiecke in immer kleinere zu unterteilen. Wenn uns dies gelingt, haben wir die Forderung der Selbstähnlichkeit erfüllt. Ein Dreieck wird folgendermaßen unterteilt: Wir berechnen die Mit-

telpunkte der Dreieckseiten. Wir erhalten drei neue Punkte. Verbinden wir diese, entstehen aus dem großen Ausgangsdreieck vier neue, kleinere. Dies reicht noch nicht, wir müssen die Elemente geringfügig verändern. Das erreichen wir, indem wir die Ecken der neu entstandenen Dreiecke immer zufällig um ein kleines Stück verschieben. Die so von uns erreichte Landschaft ist schlußendlich nicht reproduzierbar!

Um zu gewährleisten, daß die Verschiebung auch für eine große Anzahl von Iterationen unser Ausgangsmodell nicht völlig verändert und die entstehende Figur wie zerknülltes Papier erscheint, muß die Verschiebung bei weiteren Iterationen immer kleiner werden.

Bis jetzt erscheint das Vorgehen als sehr einfach, aber ein kleines Problem tritt dennoch auf. Wie gesagt, wir berechnen für jedes Dreieck drei neue Schnittpunkte. Liegen zwei Dreiecke nebeneinander, so berechnen wir den Punkt der gemeinsamen Seite doppelt. Bei allgemeinem Anlegen der Datenstruktur taucht also plötzlich ein Punkt in der Punkte-Tabelle doppelt auf. Verschieben wir nun alle Punkte der Tabelle zufällig, so wird ein und derselbe Punkt wahrscheinlich für die beiden Dreiecke unterschiedlich verschoben. Zwischen den Dreiecken klafft jetzt eine sehr unschöne Lücke. Um diesen Nachteil zu beseitigen ist es nötig, die Punkte-Tabelle nach Berechnung eines neuen Punktes zu durchsuchen, ob dieser nicht schon einmal existiert. Diese Sucherei macht die Generierung langsam.

Auch vor der Ausgabe der Landschaft als Flächenmodell empfiehlt es sich, die Dreiecke bezüglich ihrer Tiefenlage zu sortieren, damit die verdeckten Flächen nicht zu sehen sind.

Bereits bei der 3-D-Einführung wurde uns klar, daß diese Sortierung in alle drei Koordinatenrichtungen, also `x-`, `y-` und `z`-Richtung, erfolgen muß. Auch dieses Sortieren erfordert einige Zeit. Ein grundlegender Algorithmus würde also vielleicht wie folgt aussehen:

1) Eingabe der Grundstruktur und Anlage der Datenstruktur
2) `n`-maliges Durchlaufen der Unterteilung und Verschiebung:
   - Berechnung der Schnittpunkte für jedes Dreieck
   - Einfügen der neuen Punkte und Kantenzeiger in die Tabellen
   - Suchen ob ein Punkt bereits vorhanden ist
   - Zufälliges Verschieben der berechneten Punkte
3) Sortieren nach Tiefenlage
4) Ausgabe des fertigen Bildes

Daß die Programmierung dieser Routinen nicht ganz einfach ist, wird sicher jedem einleuchten. Vor allem, weil `C` keine größeren Arrays (Felder) als 65 Kbyte zuläßt (und das ist gut so, denn globale Variablenfelder werden vom Compiler direkt ans eigentliche Programm angehängt, das dann eine entsprechende Länge bekommen würde) muß mit dynamischer Speicherreservierung und Pointern gearbeitet werden, wenn hohe Unterteilung gefordert wird. Daß sich die Mühe lohnt, zeigen die Abbildungen 4-4-1-2 (1-5).

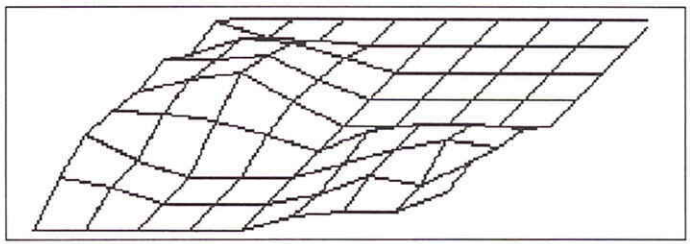
2) Verwendung eines rechteckigen Grundrasters

Auch hier gehen wir von einem Bauelement aus, dem Viereck. Wir legen uns als erstes in der `x-z`-Ebene ein rechteckiges `n*m`-Raster

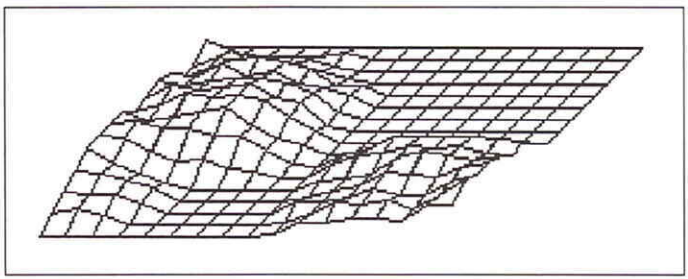
an. Jedem Rasterpunkt ordnen wir eine bestimmte Höhe zu. So kann das grundsätzliche Aussehen der Endfigur vorbestimmt werden. Ist die Initialisierung erfolgt, verdoppeln wir das Raster in x- und z-Richtung und berechnen die neuen Höhen. Jetzt verschieben wir wieder alle Punkte zufällig. Wieder wird die Verschiebung bei steigender Durchlaufzahl verringert! Allerdings dürfen wir die Punkte nur in y-Richtung verschieben, um ein »Chaos« bei der nächsten Iteration zu verhindern. Der Vorteil dieses Vorgehens liegt vor allem darin, daß wir nicht mehr sortieren müssen, denn erstens berechnen wir nun keinen Punkt mehr zweimal und außerdem ist uns die Tiefenlage der Flächen schon bekannt. Die drei Abbildungen 4-4-1-2 (1) bis (3) zeigen die Entwicklung der Landschaft bei zunehmender Unterteilung des Grundrasters als Gittermodelle.

Noch eine Anmerkung: Das Bauelement, das heißt die entstehenden Flächen, sind im eigentlichen Sinn keine Vierecke.

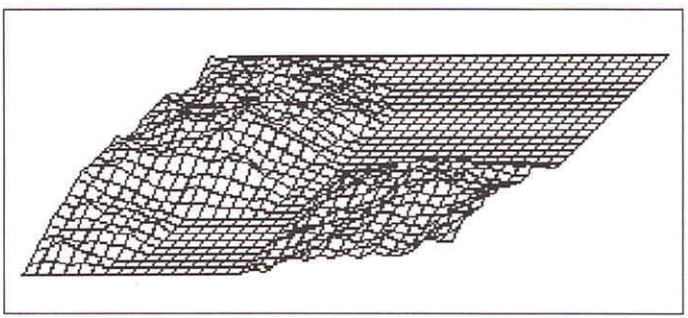
Sie bestehen zwar aus vier verbundenen Punkten, aber diese liegen nicht in einer Ebene. Deshalb muß bei einem Flächenmodell mit Beleuchtungsinterpretation diese Figur in zwei Dreiecke unterteilt werden, deren Punkte dann in einer Ebene liegen. Ist dies geschehen, kann die Farbe dieser beiden Flächen in Abhängigkeit eines Beleuchtungsvektors berechnet werden und die Ausgabe erfolgen. Aber auch diese Ausgabe hat ihre Nachteile. Sie wirkt nach einigen Durchgängen unruhig und in Teilbereichen des Bildes bilden sich regelmäßige Muster. Am besten ist es, die Farbe der beiden Dreiecke zu mitteln und dann wiederum ein Viereck zu zeichnen.



4-4-1-2 (1)



4-4-1-2 (2)



4-4-1-2 (3)

Flächenmodell einer fraktalen Landschaft (Grundelement sind Rechtecke)

Ein Algorithmus zum Erzeugen einer künstlichen Landschaft nach der zweiten Methode könnte also wie folgt aussehen:

- Grundlandschaft ins Raster einlesen
- Raster »aufblasen«, um Platz für die neuen Punkte zu schaffen
- Neue Punkte berechnen

- An allen Punkten (nur nach oben!) zufällig ein wenig zupfen
- Ausgabe der Grafik

Besonders interessant sieht das Endmodell aus, wenn die Höhe farblich interpretiert wird. Zeichnen wir zum Beispiel ab einer bestimmten Höhe alle Dreiecke mit der Farbe blau, so entsteht ein See. Die restlichen Farben (wenn wir mit 5 Planes arbeiten also 30 – zwei Register für schwarz [Hintergrund] und blau vergeben) unterteilen wir in zwei Gruppen (je 15 Farben), in eine Grün- und eine Graupalette. Nun interpretieren wir die Farbe der Dreiecke zwischen der Seehöhe und einer weiteren Höhenmarke als Grüntöne und die Flächen darüber als Grautöne. Als Resultat erhalten wir eine Alpenlandschaft mit einem See, einer Wiesen- und Waldregion und kargen Bergspitzen. Und noch etwas: Die Ausgabe wirkt erst dann richtig plastisch, wenn wir mit Perspektive arbeiten.

Zoomen wir in das fertige Landschaftsmodell, so können wir die Selbstähnlichkeit und die einzelnen Bauelemente immer wiederfinden. Außerdem erkennen wir, daß an bestimmten Stellen eine besonders hohe Unterteilung erfolgt ist. Eine Verwandtschaft zu den Bildern der komplexen rekursiven Folgen ist also sichtbar.

Zum Experimentieren steht am Ende ein kurzes Programm, mit dem nach der Methode 2 Landschaften generiert werden können. Um es möglicht kurzzuhalten, habe ich auf eine Vordefinierung der Grundform verzichtet. Ausgegangen wird von einem Raster mit 2x2 Punkten, also zwei Dreiecken.

Da ich in dem Programm mit einem Feld `land[][]` arbeite, ist die Anzahl der Iterationen auf sechs begrenzt, da »C« nur bis 65 Kbyte Felder verwaltet. Wie bereits oben angesprochen, müßte ansonsten Speicher alloziert und die gesamte Feldverwaltung selbst übernommen werden. Es wird nach einigem Experimentieren sicher auffallen, daß trotz erneutem Starten des Programms immer wieder die gleichen Landschaften entstehen. Dies liegt an der Funktion `RangeRand()`, die (leider) nicht wirklich zufällige Werte produziert. Wie bei dem Programm *fractal.c* habe ich auch hier die notwendigen Vordefinitionen für die Menüs vom eigentlichen Programm abgetrennt. Das File *land.menu* muß also im Hautprogramm mit `#include` eingebunden werden.

Das Programm bietet folgende Möglichkeiten:

- »*Projekt* Menü«
- »*Save*«
  Speichern des Bildes im gepackten IFF-Format
- »*Ende*«
  Beendet das Programm
- »*Start* Menü«
- »Ausgabe«
- »*Fläche*«
  gibt das Feld in 3-D-Form mit gefüllten Flächen aus.
- »*Gitter*«
  gibt das Feld in 3-D-Form als Gitternetz aus.
- »*Iteration*«
  eine neue Iteration wird durchlaufen. Der Aufruf beinhaltet »Aufblasen« des Feldes, berechnen der neuen Feldelemente und zufällige Verschiebung. Nach der Berechnung wird das neue Feld im gewählten

Modus entweder als Flächenmodell oder als Gitternetz ausgegeben.

- *»Neu Starten«*
  setzt alle Feldelemente auf Null und startet neu, denn wir wollen ja viele verschiedene Landschaften erzeugen.

Programm: *land.menu (Include File)*

```
/* Include File land.menu fuer das Programm land.c */


#define WIDTH   320  /* Screen Modus */
#define HEIGHT  200
#define DEPTH   5
#define FLAGS   0
#define FWIDTH  200  /* Groesse der Grundflaeche */
#define FHEIGHT 100
#define FXOFF   120
#define FYOFF   100

#define NMAX 130    /* max. Anzahl der Stuetzpunkte in x,y Richtung */
#define N0 2        /*Anzahl der Stuetzpunkte in x,y Richtung       */
#define BEREICH 100 /* Bereich fuer zufaellige Verschiebung         */
#define OFF 40      /* Verschiebung in negative Richtung            */
#define MEER 0      /* Meereshoehe                                  */
#define HOEHE 30    /* Hoehe ab der Grau-Palette verwendet wird     */

extern short save();


/* Menue 2 */
struct IntuiText Text1={3,1,COMPLEMENT,1,1,0,"Neu starten",0};
struct MenuItem MenuItem3=
 {0,0,22,129,10,ITEMTEXT+COMMSEQ+ITEMENABLED+HIGHCOMP,
  0,&Text1,0,'N',0,0xFFFF};
struct IntuiText Text2={3,1,COMPLEMENT,1,1,0,"Iteration",0};
struct MenuItem MenuItem2=
 {&MenuItem3,0,11,129,10,ITEMTEXT+COMMSEQ+ITEMENABLED+HIGHCOMP,
  0,&Text2,0,'I',0,0xFFFF};
struct IntuiText Text3={3,1,COMPLEMENT,1,1,0,"Gitter",0};
struct MenuItem SubItem2=
 {0,114,2,97,10,ITEMTEXT+COMMSEQ+ITEMENABLED+HIGHCOMP,
  0,&Text3,0,'G',0,0xFFFF};
struct IntuiText Text4={3,1,COMPLEMENT,1,1,0,"Flaeche",0};
struct MenuItem SubItem1=
 {&SubItem2,114,-8,97,10,ITEMTEXT+COMMSEQ+ITEMENABLED+HIGHCOMP,
  0,&Text4,0,'F',0,0xFFFF};
struct IntuiText Text5={3,1,COMPLEMENT,1,1,0,"Ausgabe",0};
struct MenuItem MenuItem1=
 {&MenuItem2,0,0,129,10,ITEMTEXT+ITEMENABLED+HIGHCOMP,
  0,&Text5,0,0,&SubItem1,0xFFFF};
struct Menu Menu2={0,81,0,57,0,MENUENABLED,"Start",&MenuItem1};


/* Menue 1 */
struct IntuiText Text6={3,1,COMPLEMENT,1,1,0,"Ende",0};
struct MenuItem MenuItem5=
 {0,0,11,73,10,ITEMTEXT+COMMSEQ+ITEMENABLED+HIGHCOMP,
  0,&Text6,0,'E',0,0xFFFF};
struct IntuiText Text7={3,1,COMPLEMENT,1,1,0,"Save",0};
struct MenuItem MenuItem4=
 {&MenuItem5,0,0,73,10,ITEMTEXT+COMMSEQ+ITEMENABLED+HIGHCOMP,
  0,&Text7,0,'S',0,0xFFFF};
struct Menu Menu1={&Menu2,0,0,75,0,MENUENABLED,"Projekt",&MenuItem4};


struct IntuiMessage *MyMessage;
struct IntuitionBase *IntuitionBase;
struct GfxBase *GfxBase;
struct Screen *MyScreen;
struct Window *MyWindow;
struct RastPort *MyRastPort;
struct AreaInfo MyAreaInfo;
struct TmpRas MyTmpRas;
struct NewScreen MyNewScreen=
 {0,0,WIDTH,HEIGHT,DEPTH,1,0,FLAGS,CUSTOMSCREEN,0,0,0,0};
struct NewWindow MyNewWindow=
 {0,0,WIDTH,HEIGHT,0,1,MENUPICK,NOCAREREFRESH+BORDERLESS+
  ACTIVATE,0,0,0,0,0,0,0,0,0,CUSTOMSCREEN};
```

Programm: *land.c*

```
/*********************************************************************/
/* land.c                                              15.Ø1.88      */
/* Version 2.Ø                                                       */
/*                                                                   */
/* Generierung nach Verfahren 2                                      */
/* Als  Initialisierung wird von einer ebenen Flaeche ausgegangen    */
/* (alle Rasterelemente Null!).                                      */
/*                                                                   */
/*                                                                   */
/* fuer Markt+Technik geschrieben von Heiko Knappe                   */
/*                                                                   */
/* Komplexe Dynamische Systeme                                       */
/*********************************************************************/


#include <exec/types.h>
#include <math.h>
#include <intuition/intuition.h>
#include <graphics/gfxmacros.h>
#include "land.menu"


int   n=NØ,nØ,iter,iØ,jØ;    /* Wieviel Unterteilungen?  */
int   i,j,k;
short land[NMAX][NMAX];      /* Feld mit den Punktwerten */
short x[4],y[4];             /* 4 Eckpunkte der Flaechen */
short stx,sty;
short col1,col2;
UWORD buffer[5*1Ø];          /* Variablen fuer AreaFill  */
long  *mem;
short bx=1,by=1,bz=1;        /* Beleuchtungsvektor       */
float b;
char  mode=Ø;                /* Zeichenmodus             */


/***********************/
/* Hauptprogramm       */
/***********************/

main(argc,argv)
int argc;
char *argv[];
{
 open_things();
 ausgabe();
 for(;;)
 {
    while((MyMessage=GetMsg(MyWindow->UserPort))==Ø);
    ReplyMsg(MyMessage);
    switch(MyMessage->Class)
    {
       case CLOSEWINDOW: close_things();
       case MENUPICK:    do_menu();
                         break;
    }
 }
}

do_menu()
{
 short menu=MENUNUM(MyMessage->Code),item=ITEMNUM(MyMessage->Code);
 short subitem=SUBNUM(MyMessage->Code);

 switch(menu)
 {
    case Ø: /* Projekt Menu */
       switch(item)
       {
          case Ø: /* Save */
             ModifyIDCMP(MyWindow,MENUPICK+MENUVERIFY);
             save("land.pic",1,MyScreen);
             ModifyIDCMP(MyWindow,MENUPICK);
             break;
          case 1: /* Ende */
             close_things();
       }
       break;
    case 1: /* Start Menu */
       switch(item)
```

```
80        {
81           case Ø: /* Ausgabe */
82              switch(subitem)
83              {
84                 case Ø: /* Flaechen */
85                    mode=Ø;
86                    ausgabe();
87                    break;
88                 case 1: /* Gitter */
89                    mode=1;
90                    ausgabe();
91                    break;
92              }
93              break;
94           case 1: /* naechste Iteration */
95              next_iteration();
96              break;
97           case 2: /* neu Starten */
98              n=NØ; iter=Ø;
99              for(i=Ø;i<NMAX;++i)
100                for(j=Ø;j<NMAX;++j)
101                   land[i][j]=Ø;
102             ausgabe();
103             break;
104       }
105       break;
106    }
107 }
108
109
110 /************************/
111 /* Initialisierung     */
112 /************************/
113
114 open_things()
115 {
116  if((IntuitionBase=OpenLibrary("intuition.library",Ø))==Ø)
117     close_things();
118  if((GfxBase=OpenLibrary("graphics.library",Ø))==Ø)
119     close_things();
120  if((MyScreen=OpenScreen(&MyNewScreen))==Ø)
121     close_things();
122  MyNewWindow.Screen=MyScreen;
123  if((MyWindow=OpenWindow(&MyNewWindow))==Ø)
124     close_things();
125  SetMenuStrip(MyWindow,&Menu1);
126  MyRastPort=MyWindow->RPort;
127  init_colors();
128  init_fill();
129  b=sqrt((float)(bx*bx+by*by+bz*bz));
130 }
131
132 init_fill()
133 {
134  if((mem=AllocRaster(WIDTH,HEIGHT))==Ø)
135     close_things();
136  MyRastPort->AreaInfo=&MyAreaInfo;
137  MyRastPort->TmpRas=InitTmpRas(&MyTmpRas,mem,RASSIZE(WIDTH,HEIGHT));
138  InitArea(&MyAreaInfo,&buffer[Ø],1Ø);
139 }
140
141 close_things()
142 {
143  if(MyWindow!=Ø) CloseWindow(MyWindow);
144  if(MyScreen!=Ø) CloseScreen(MyScreen);
145  if(IntuitionBase!=Ø) CloseLibrary(IntuitionBase);
146  if(GfxBase!=Ø) CloseLibrary(GfxBase);
147  if(mem!=Ø) FreeRaster(mem,WIDTH,HEIGHT);
148  exit(Ø);
149 }
150
151 init_colors()
152 {
153  SetRGB4(&(MyScreen->ViewPort),Ø,Ø,Ø,Ø);   /* Hintergrund schwarz */
154  for(i=1;i<16;++i)                         /* Gruene Palette      */
155     SetRGB4(&(MyScreen->ViewPort),i,Ø,i,Ø);
156  for(i=16;i<31;++i)                        /* Graue Palette       */
157     SetRGB4(&(MyScreen->ViewPort),i,i-15,i-15,i-15);
158  SetRGB4(&(MyScreen->ViewPort),31,Ø,Ø,15); /* Blau */
159 }
160
```

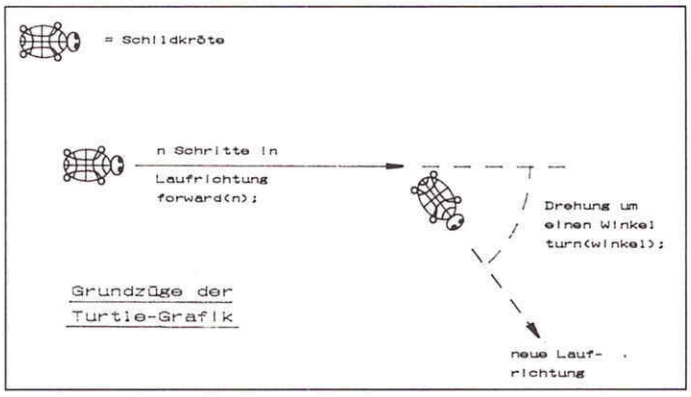
```
161
162 /************************/
163 /* Berechnung der      */
164 /* Stuetzpunkte        */
165 /************************/
166
167 next_iteration()
168 {
169  if(n>=NMAX-(n-1)) return(Ø);
170  ++iter;
171  aufblasen();
172  neu_berechnen();
173  verschieben();
174  ausgabe();
175 }
176
177 aufblasen()
178 /* bisheriges Array aufblasen um Platz fuer die neuen Punkte zu */
179 /* schaffen.                                                    */
180 {
181  nØ=n-1;
182  n+=(n-1);
183  for(iØ=nØ,i=n-1;i>=Ø;i-=2,--iØ)
184     for(jØ=nØ,j=n-1;j>=Ø;j-=2,--jØ)
185        land[i][j]=land[iØ][jØ];
186 }
187
188 neu_berechnen()
189 {
190  for(i=1;i<=n;i+=2)
191     for(j=1;j<=n;j+=2)
192     {
193        land[i][j-1]=(land[i-1][j-1]+land[i+1][j-1])/2;
194        land[i-1][j]=(land[i-1][j-1]+land[i-1][j+1])/2;
195     }
196  for(i=1;i<n;i+=2)
197     for(j=1;j<n;j+=2)
198        land[i][j]=(land[i-1][j]+land[i+1][j])/2;
199 }
200
```

```
201 verschieben()
202 {
203  for(i=Ø;i<n;++i) /* in x-Richtung */
204     for(j=Ø;j<n;++j) /* in y-Richtung */
205     {
206        if(land[i][j]>=MEER)
207           land[i][j]+=(RangeRand(BEREICH)-OFF)/iter/iter;
208        if(land[i][j]<MEER) land[i][j]=MEER-1; /* unter dem Meer */
209        /* zufaellige Verschiebung unter Beruecksichtigung der   */
210        /* Anzahl der Iterationen.                               */
211     }
212 }
213
214
215 /************************/
216 /* Ausgabe             */
217 /************************/
218
219 ausgabe()
220 {
221  SetRast(MyRastPort,Ø);
222  stx=FWIDTH/n,sty=FHEIGHT/n;
223  for(i=Ø;i<n-1;++i)
224  {
225     for(j=Ø;j<n-1;++j)
226     {
227        /* Berechnung der Eckpunkte der Flaechen */
228        x[Ø]=FXOFF+stx*i-sty*j,y[Ø]=FYOFF-land[i][j]+sty*j;
229        x[1]=FXOFF+stx*(i+1)-sty*j,y[1]=FYOFF-land[i+1][j]+sty*j;
230        x[2]=FXOFF+stx*(i+1)-
           sty*(j+1),y[2]=FYOFF-land[i+1][j+1]+sty*(j+1);
231        x[3]=FXOFF+stx*i-sty*(j+1),y[3]=FYOFF-land[i][j+1]+sty*(j+1);
232        for(k=Ø;k<=3;++k)
233        {
234           if(y[k]<Ø) y[k]=Ø;
235           if(y[k]>=HEIGHT) y[k]=HEIGHT-1;
236        }
237        zeich();
238     }
239  }
240 }
```

```
241
242 zeich()
243 {
244  if(mode==Ø)
245  {
246     farbe();
247     SetOPen(MyRastPort,col1);
248     SetAPen(MyRastPort,col1);  /* Flaeche1 zeichnen */
249     AreaMove(MyRastPort,x[Ø],y[Ø]); AreaDraw(MyRastPort,x[1],y[1]);
25Ø     AreaDraw(MyRastPort,x[2],y[2]); AreaEnd(MyRastPort);
251     SetOPen(MyRastPort,col2);
252     SetAPen(MyRastPort,col2);  /* Flaeche2 zeichnen */
253     AreaMove(MyRastPort,x[Ø],y[Ø]); AreaDraw(MyRastPort,x[3],y[3]);
254     AreaDraw(MyRastPort,x[2],y[2]); AreaEnd(MyRastPort);
255  }
256  else
257  {
258     SetOPen(MyRastPort,31); SetAPen(MyRastPort,Ø);
259     AreaMove(MyRastPort,x[Ø],y[Ø]); AreaDraw(MyRastPort,x[1],y[1]);
26Ø     AreaDraw(MyRastPort,x[2],y[2]); AreaDraw(MyRastPort,x[3],y[3]);
261     AreaEnd(MyRastPort);
262  }
263 }
264
265 farbe()  /* Berechnung der Farbe einer Flaeche */
266 {
267  float n,h;
268  int nx,ny,nz;
269  short v1x,v1y,v1z,v2x,v2y,v2z;
27Ø
271  /* 1. Flaeche */
272  if(land[i][j]<MEER&&land[i+1][j]<MEER&&land[i+1][j+1]<MEER)
273     col1=31;
274  else
275  {
276     v1x=1,v1y=Ø,v1z=land[i+1][j]-land[i][j];
277     v2x=Ø,v2y=1,v2z=land[i+1][j+1]-land[i][j];
278     nx=v1y*v2z-v1z*v2y,ny=v1z*v2x-v1x*v2z,nz=v1x*v2y-v1y*v2x;
279     n=sqrt((float)(nx*nx+ny*ny+nz*nz));
28Ø     if(n==Ø) col1=1;
281     else
282     {
283        h=acos((float)(nx*bx+ny*by+nz*bz)/b/n)-PI/2;
284        h=(h<Ø)? -h : h;
285        if((col1=(short)((PI/2-h)*15*2/PI)+1)>15) col1=15;
286        if(col1<=Ø) col1=1;
287     }
288     if(land[i][j]>=HOEHE&&land[i+1][j]>=HOEHE&&land[i+1][j+1]>=HOEHE)
289        col1+=15;
29Ø  }
291  /* 2. Flaeche */
292  if(land[i][j]<MEER&&land[i][j+1]<MEER&&land[i+1][j+1]<MEER)
293     col2=31;
294  else
295  {
296     v1x=1,v1y=Ø,v1z=land[i][j+1]-land[i][j];
297     v2x=Ø,v2y=1,v2z=land[i+1][j+1]-land[i][j];
298     nx=v1y*v2z-v1z*v2y,ny=v1z*v2x-v1x*v2z,nz=v1x*v2y-v1y*v2x;
299     n=sqrt((float)(nx*nx+ny*ny+nz*nz));
3ØØ     if(n==Ø) col2=1;
3Ø1     else
3Ø2     {
3Ø3        h=acos((float)(nx*bx+ny*by+nz*bz)/b/n)-PI/2;
3Ø4        h=(h<Ø)? -h : h;
3Ø5        if((col2=(short)((PI/2-h)*15*2/PI)+1)>15) col2=15;
3Ø6        if(col2<=Ø) col2=1;
3Ø7     }
3Ø8     if(land[i][j]>=HOEHE&&land[i][j+1]>=HOEHE&&land[i+1][j+1]>=HOEHE)
3Ø9        col2+=15;
31Ø  }
311 }
```

4.4.2 Fraktale Computergrafiken – Turtle-Grafik

Bisher haben wir fraktale Grafiken nur in der Form von rekursiven komplexen Folgen kennengelernt und untersucht. Wir erkannten, daß die wichtigsten Elemente dieser Erscheinungen die Selbstähnlichkeit und die Rückkopplung waren. Es ist klar, daß wir nicht



4-4-2-1 Grundzüge der Turtle-Grafik

nur mit rekursiven Folgen eine Wirkung des vorherigen auf den augenblicklichen Wert erreichen können. Als Ausblick soll zum Abschluß des Kapitels – Der Vorstoß ins Chaos – noch eine ganz andere Art der fraktalen Computergrafik stehen.

Das Stichwort ist hier Schildkrötengrafik (Turtle-Grafik). Sie kommt aus der Sprache Logo und ist wohl den meisten von uns ein Begriff. Man positioniert die Schildkröte auf einem bestimmten Bildpunkt und läßt sie dann in eine Richtung eine bestimmte Strecke entweder vorwärts (`forward()`) oder rückwärts (`backward()`) laufen. Bei der Fortbewegung hinterläßt sie dann eine sichtbare Spur. Außer vor- bzw. zurücklaufen kann sie sich auch noch um einen bestimmten Winkel drehen (`turn()`) (siehe Abbildung 4-4-2-1). So kann man also jeden Bildpunkt erreichen! Folgende Funktionen benötigen wir also: `forward(n);` `backward(n);` bewegt die Schildkröte in ihre augenblickliche Richtung um `n` Schritte, `turn(ß);` dreht die Laufrichtung um `ß` Grad im Uhrzeigersinn.

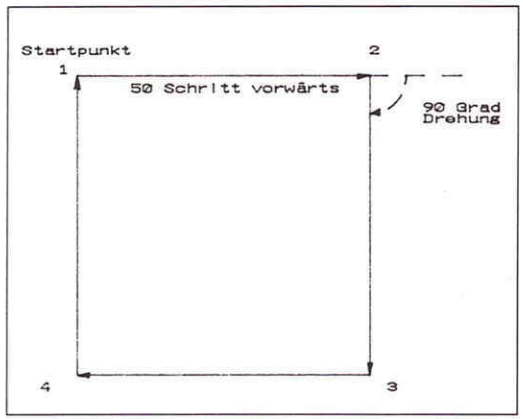
Diese Routinen werden folgendermaßen realisiert:

```
forward(step)
short step;
{
 x+=step*cos((float)winkel*PI/180);
 y+=step*sin((float)winkel*PI/180);
 if(x>=0&&x<WIDTH&&y>=0&&y<HEIGHT) Draw(rp,x,y);
}

backward(step)
short step;
{
 forward(-step);
}

turn(a)
short a;
{
 winkel=(winkel+a)%360;
}
```

So, nun versuchen wir doch einmal ein Quadrat mit der Kantenlänge 50 zu ziehen (Die Schildkröte ist bereits auf die linke obere Ecke des zu zeichnenden Quadrats positioniert worden und ihre Laufrichtung zeigt nach links.):

```
forward(50); turn(90);
forward(50); turn(90);
forward(50); turn(90);
forward(50); turn(90);   (siehe Abbildung 4-4-2-2)
```

Wir gehen also viermal um 50 Schritte vorwärts und drehen dann immer um 90° Grad. Es ist klar, daß wir dann wieder den Ausgangspunkt erreicht haben und das Quadrat gezeichnet ist.
Erstaunlicher wird es aber, wenn wir kompliziertere Entstehungsalgorithmen zugrundelegen. Dann entstehen die bekannten Hilbert-, Koch- und Drachen-Kurven. Wesentlichstes Merkmal ist wie gesagt auch bei Ihnen die



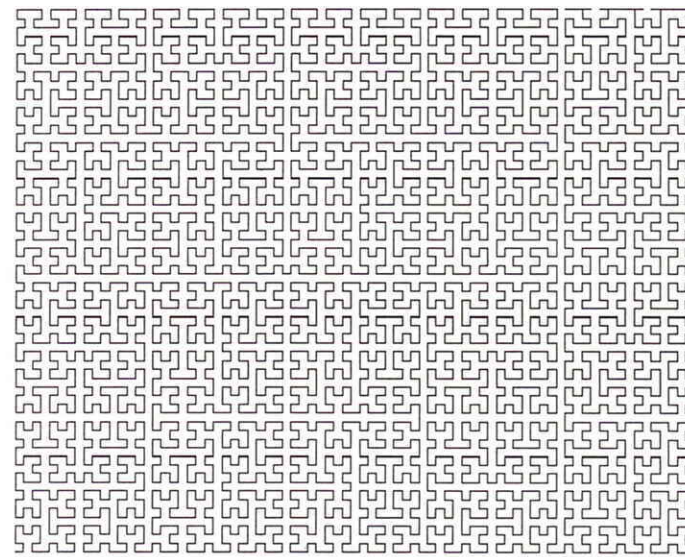
4-4-2-2

Rückwirkung des augenblicklichen Zustands auf den nächsten. Sie wird durch mehrfaches Aufrufen des gleichen Algorithmus erreicht. Hier noch eine Warnung: Mehrfaches Aufrufen einer Funktion bedeutet auf Dauer ein Überlaufen des Stacks, da bei jedem Ansprung ohne Rücksprung die lokalen Variablen neu auf den Stack gelegt werden. Sollten hier Probleme auftreten, so ist der Stack entprechend mit `stack SIZE` im CLI hochzusetzen.

Die Hilbert-Kurve

Als erstes wollen wir die sogenannte Hilbert-Kurve generieren (siehe Abbildung 4-4-2-3). Betrachtet man die fertige Kurve, so ist der Entstehungsalgorithmus sicher nicht unmittelbar zu erkennen. Auch umgekehrt ist es kaum möglich aus der iterativen Routine auf das Aussehen der fertigen Kurve zu schließen. Wir entdecken also wieder wie kompliziert es ist, das Ergebnis einer Rekursion vorherzusehen. Auch ist die Selbstähnlichkeit gut zu erkennen.

Von dieser Art der Entstehung, die zwar etwas völlig anderes ist als die rekursiven



4-4-2-3 Die Hilbert-Kurve

Folgen, erhalten wir trotzdem ähnliche Phänomene und Kurven erstaunlicher Ordnung und Schönheit. Welche Art der Grafik schöner aussieht, ist und bleibt schlußendlich eine persönliche Entscheidung.

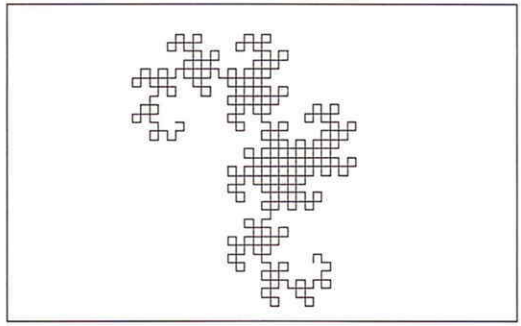
Programm: *hilbert.c*

```
 1 /*************************************************************************/
 2 /* hilbert.c                                              15.Ø1.88   */
 3 /* Version 1.Ø                                                       */
 4 /*                                                                   */
 5 /* Hilbert-Kurve                                                     */
 6 /*                                                                   */
 7 /*                                                                   */
 8 /* fuer Markt+Technik geschrieben von Heiko Knappe                   */
 9 /*                                                                   */
1Ø /* Komplexe Dynamische Systeme                                       */
11 /*************************************************************************/
12
13 #include <math.h>
14 #include <exec/types.h>
15 #include <intuition/intuition.h>
```

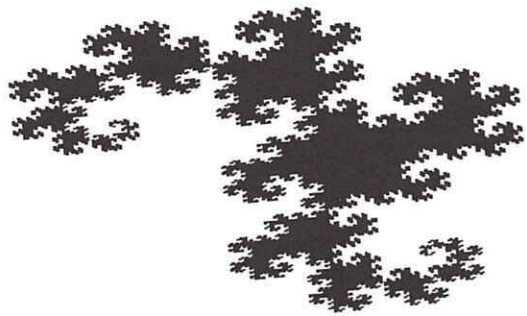
```
16
17 #define WIDTH  64Ø          /* Screen Modus */
18 #define HEIGHT 5ØØ
19 #define DEPTH  1
2Ø #define FLAGS  HIRES+LACE
21
22 #define PI 3.1415
23
24 #define SEITE 8           /* Startwerte */
25 #define STARTWINKEL Ø
26 #define GRAD  1Ø
27 #define RICHTUNG 1
28 #define XSTART Ø
29 #define YSTART HEIGHT-1
3Ø
31
32 extern short save();
33
34 struct IntuitionBase *IntuitionBase;
35 struct GfxBase *GfxBase;
36 struct Screen *MyScreen;
37 struct RastPort *rp;
38 struct NewScreen MyNewScreen=
39  {Ø,Ø,WIDTH,HEIGHT,DEPTH,1,Ø,FLAGS,CUSTOMSCREEN,Ø,Ø,Ø,Ø};
4Ø
41 short x=XSTART,y=YSTART,winkel=STARTWINKEL;
42
43
44 /***********************/
45 /* Hauptprogramm       */
46 /***********************/
47
48 main(argc,argv)
49 int argc;
5Ø char *argv[];
51 {
52  open_things();
53  hilbert(GRAD,SEITE,RICHTUNG);
54  save("Hilbert.pic",1,MyScreen);
55  close_things();
56 }
57
58
59 /***********************/
6Ø /* Initialisierung     */
61 /***********************/
62
63 open_things()
64 {
65  if((IntuitionBase=OpenLibrary("intuition.library",Ø))==Ø)
66     close_things();
67  if((GfxBase=OpenLibrary("graphics.library",Ø))==Ø)
68     close_things();
69  if((MyScreen=OpenScreen(&MyNewScreen))==Ø)
7Ø     close_things();
71  rp=&(MyScreen->RastPort);
72  Move(rp,x,y);
73  SetRGB4(&(MyScreen->ViewPort),Ø,Ø,Ø,Ø);
74  SetRGB4(&(MyScreen->ViewPort),1,15,Ø,Ø);
75 }
76
77 close_things()
78 {
79  if(IntuitionBase!=Ø) CloseLibrary(IntuitionBase);
8Ø  if(GfxBase!=Ø) CloseLibrary(GfxBase);
81  if(MyScreen!=Ø) CloseScreen(MyScreen);
82  exit();
83 }
84
85
86 /***********************/
87 /* Berechnung          */
88 /***********************/
89
9Ø hilbert(grad,seite,richtung)
91 short grad,seite;
92 BYTE  richtung;
93 {
94  if(MyScreen->TopEdge>1ØØ) return(1);
95  if(grad>Ø)
```

```
 96  {
 97     turn(-richtung*9Ø); hilbert(grad-1,seite,-richtung); forward(seite);
 98     turn(richtung*9Ø); hilbert(grad-1,seite,richtung); forward(seite);
 99     hilbert(grad-1,seite,richtung); turn(richtung*9Ø); forward(seite);
1ØØ     hilbert(grad-1,seite,-richtung); turn(-richtung*9Ø);
1Ø1  }
1Ø2 }
1Ø3
1Ø4 forward(step)
1Ø5 short step;
1Ø6 {
1Ø7  x+=step*cos((float)winkel*PI/18Ø);
1Ø8  y+=step*sin((float)winkel*PI/18Ø);
1Ø9  if(x>=Ø&&x<WIDTH&&y>=Ø&&y<HEIGHT) Draw(rp,x,y);
11Ø }
111
112 backward(step)
113 short step;
114 {
115  forward(-step);
116 }
117
118 turn(a)
119 short a;
12Ø {
121  winkel=(winkel+a)%36Ø;
122 }
```

Da die gesamte Initialisierung und die Turtle-Routinen gleichbleiben, werden im folgenden nur noch die eigentliche Rechenroutine (im Falle von *hilbert.c* die Funktion `hilbert();`) und die Startbedingungen angegeben.



4-4-2-4 (1) Die Drachen-Kurve



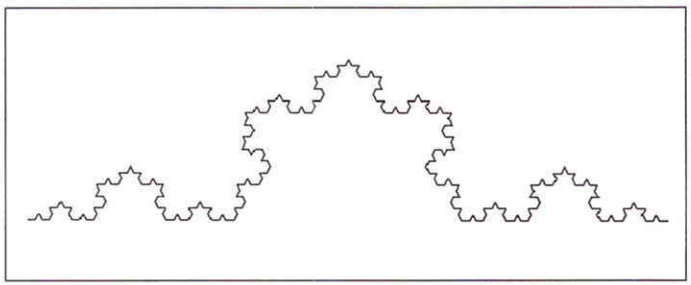
4-4-2-4 (2) Drachenkurve höheren Grades

Die Drachen-Kurve

(siehe Abbildung 4-4-2-4)

```
#define SEITE 8          /* Startwerte */
#define STARTWINKEL Ø
#define GRAD  9
#define XSTART 2ØØ
#define YSTART 2ØØ

drachen(grad,seite)
short grad,seite;
{
 if(MyScreen->TopEdge>1ØØ) return(1);
 if(grad==Ø)
 {
    forward(seite);
 }
 else
 {
```

```
    if(grad>Ø)
    {
       drachen(grad-1,seite);
       turn(9Ø);
       drachen(-(grad-1),seite);
    }
    else
    {
       drachen(-(grad+1),seite);
       turn(27Ø);
       drachen(grad+1,seite);
    }
 }
}
```



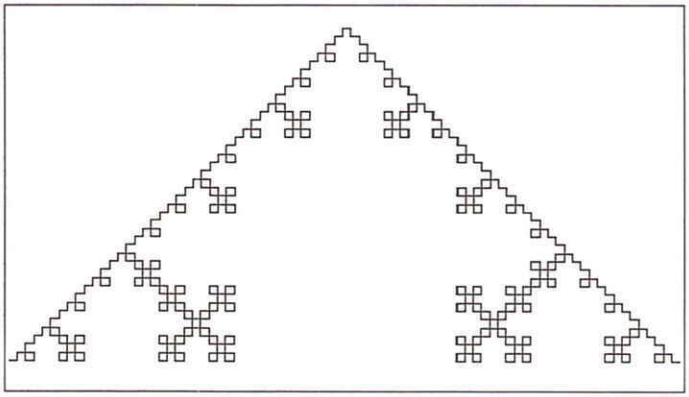
4-4-2-5 Koch-Kurve

Koch-Kurven

(1) (siehe Abbildung 4-4-2-5)

```
#define SEITE 5ØØ          /* Startwerte */
#define STARTWINKEL Ø
#define GRAD  4
#define XSTART Ø
#define YSTART 2ØØ

koch(grad,seite)
short grad,seite;
{
```

```
 if(MyScreen->TopEdge>1ØØ) return(1);
 if(grad==Ø)
 {
    forward(seite);
 }
 else
 {
    koch(grad-1,seite/3);
    turn(-6Ø);
    koch(grad-1,seite/3);
    turn(12Ø);
    koch(grad-1,seite/3);
    turn(-6Ø);
    koch(grad-1,seite/3);
 }
}
```



4-4-2-6 Quadratische Koch-Kurve

(2) (siehe Abbildung 4-4-2-6)

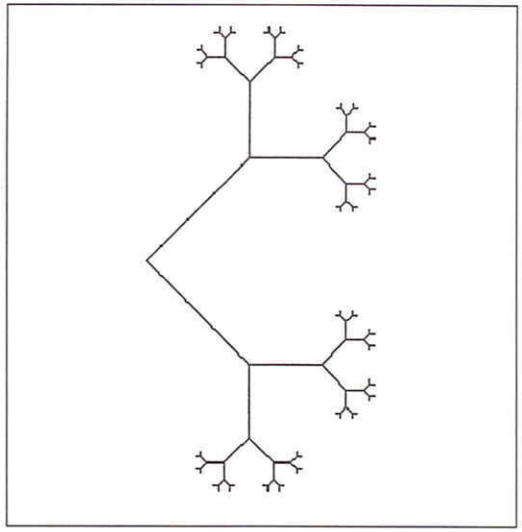
```
#define SEITE 5ØØ          /* Startwerte */
#define STARTWINKEL Ø
#define GRAD  4
#define XSTART 5Ø
#define YSTART 4ØØ

koch(grad,seite)
short grad,seite;
```

```
{if(MyScreen->TopEdge>1ØØ) return(1);
 if(grad==Ø)
 {
    forward(seite);
 }
 else
 {
    koch(grad-1,seite/3);
    turn(-9Ø);
    koch(grad-1,seite/3);
    turn(9Ø);
    koch(grad-1,seite/3);
    turn(9Ø);
    koch(grad-1,seite/3);
    turn(-9Ø);
    koch(grad-1,seite/3);
 }
}
```

Baum-Kurven

(siehe Abbildung 4-4-2-7)

```
#define SEITE 5Ø          /* Startwerte */
#define STARTWINKEL Ø
#define GRAD  5
#define XSTART 2ØØ
#define YSTART 2ØØ

tree(grad,seite)
short grad,seite;
{
 if(MyScreen->TopEdge>1ØØ) return(1);
 if(grad>Ø)
 {
    turn(-45); forward(seite); tree(grad-1,seite/2); backward(seite);
    turn(9Ø); forward(seite); tree(grad-1,seite/2); backward(seite);
    turn(-45);
 }
}
```



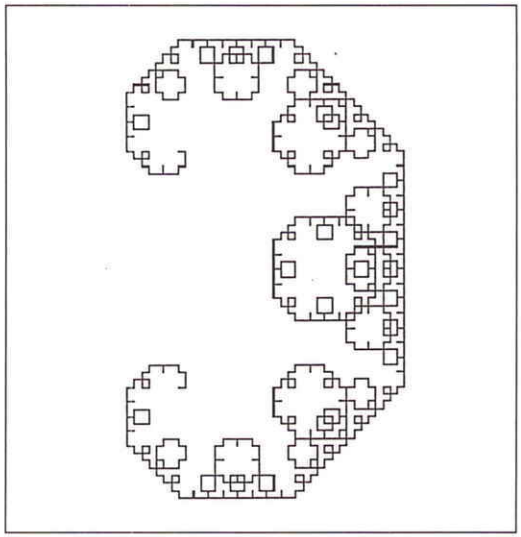
4-4-2-7 Baum-Kurve

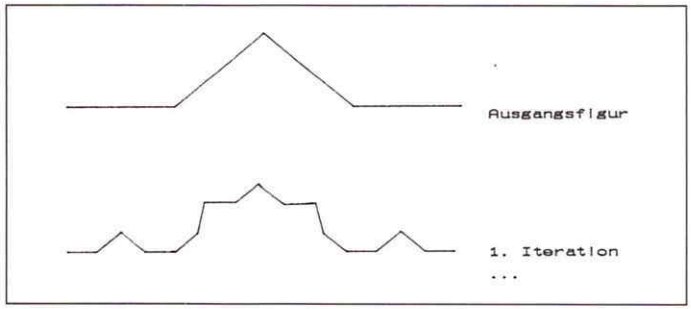
Die C-Kurve

(siehe Abbildung 4-4-2-8)

```
#define SEITE 5          /* Startwerte */
#define STARTWINKEL Ø
#define GRAD  1Ø
#define XSTART 2ØØ
#define YSTART 2ØØ

c_(grad,seite)
short grad,seite;
{
 if(MyScreen->TopEdge>1ØØ) return(1);
 if(grad==Ø)
 {
    forward(seite);
 }
 else
 {
    c_(grad-1,seite); turn(9Ø);
    c_(grad-1,seite); turn(-9Ø);
 }
}
```

4-4-2-8
C-Kurve





4-4-2-9 Entstehung der Koch-Kurve

Die oben aufgeführten Entstehungsprozesse sind natürlich nur einige aus einer Vielzahl von möglichen. Am Anfang sollte man etwas mit den Startparametern »spielen«, um so die Eigenschaften der Rekursion verstehen zu können. Dann ist es der nächste Schritt mit eigenen Iterationen zu arbeiten und neue Kurven zu entwickeln. Auch hier ist es möglich eigene, ganz neue Grafiken zu entdecken – gehen Sie deshalb auch hier auf Entdeckungsreise.

Zum Abschluß noch einige Anmerkungen

- Besonders schöne Effekte ergibt auch die Verwendung von zufälligen Änderungen während des Durchlaufs des Algorithmus. Ersetzen Sie doch zum Beispiel einmal eine Drehung um 45° durch `45°+RangeRand(1Ø°)`. (`RangeRand(n);` ist eine Betriebssystem-Routine, die eine »zufällige« Zahl zwischen `Ø` und `n` liefert. Sie ist jedoch nur scheinbar zufällig, denn bei neuem Starten des Programms werden die gleichen Zufallsfolgen generiert.) Die nun entstehenden Bilder sind einmalig und nicht unbedingt reproduzierbar!

- Die Entstehung zum Beispiel der Koch-Kurve (1) kann man außer durch das Durchlaufen der rekursiven Formel auch anders generieren:
  Wir gehen von einem Streckenzug gemäß Abbildung 4-4-2-9 aus. In jedem Durchgang ersetzen wir jede Seite durch einen Streckenzug der gleichen Form und der entsprechenden Größe. Wie wir uns vorstellen können, entsteht nach einigen Durchgängen die schon bekannte Koch-Kurve. (Gehen wir von einem Dreieck aus und führen obigen Algorithmus mehrere Male hintereinander aus, so entsteht eine Figur, die erstaunliche Ähnlichkeit mit einem Schneekristall hat.)
  Auch die Drachen-Kurve kann durch ein solches ständiges Ersetzen der Seiten des Streckenzugs erzeugt werden.

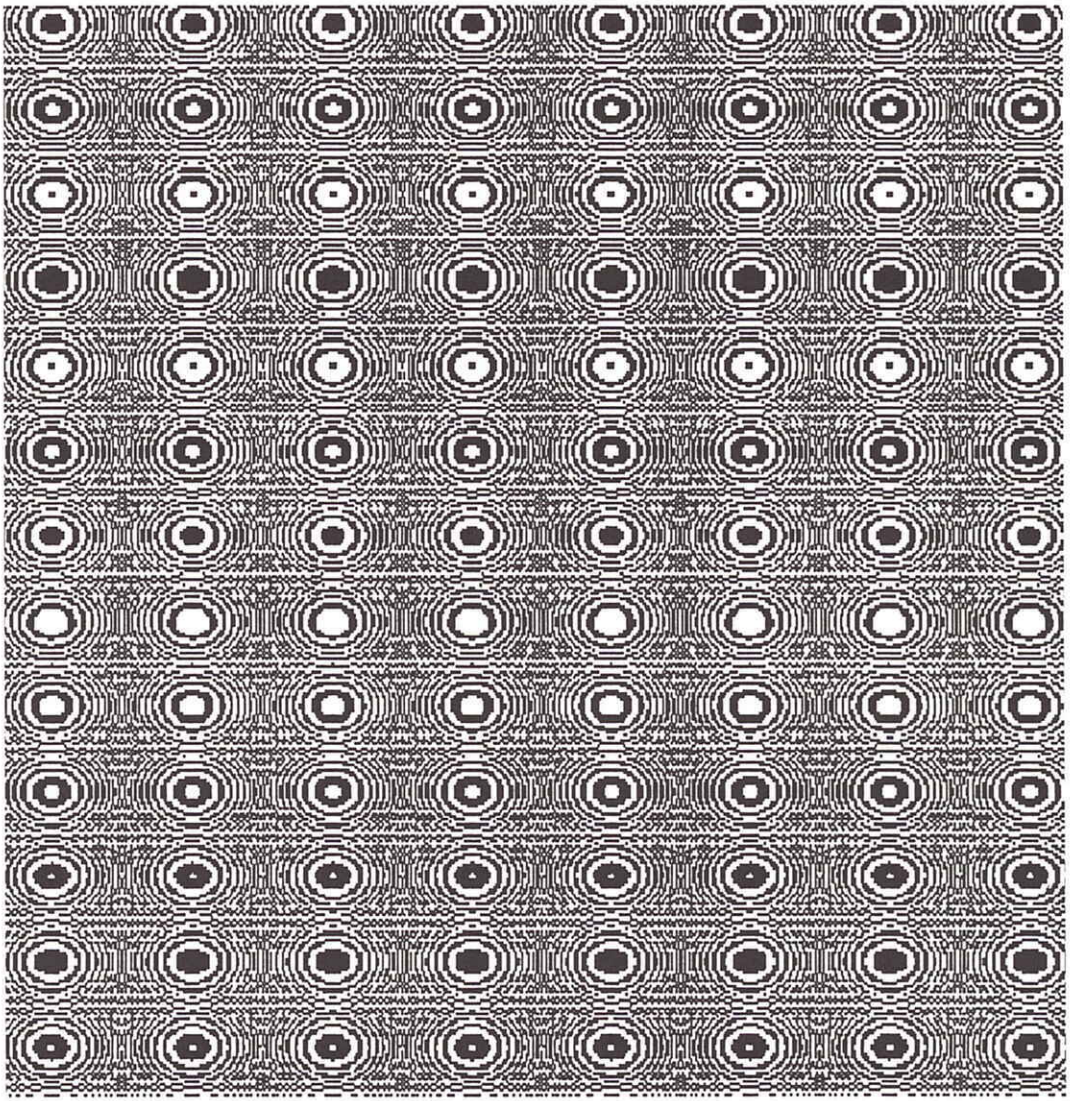
4.4.3 Tapetenmuster

Wir kennen mittlerweile die Mandelbrot-Formel zur Genüge. Wie man mit ähnlichen Abbildungsvorschriften und kleinen Veränderungen zu unendlich variationsreichen Mustern kommt, sollen die folgenden Beispiele dokumentieren:

1) Variationen des Kreises

Wie bei den komplexen Systemen, weisen wir jedem Punkt des Bildschirms relative Koordinaten zu. Jeder Punkt erhält nun nach folgender Abbildungsvorschrift eine Farbe zugewiesen:

- Wir berechnen den Abstand des Punktes (x,y) vom Urpsrung zum Quadrat, also: $(x^2 + y^2)$ und schneiden die Stellen hinter dem Komma ab.
- Ist diese Zahl durch 2 teilbar, so zeichnen wir den Punkt schwarz, ansonsten weiß (Modulo 2).



4-4-3-1 Funktion x^2+y^2

Diese einfache Vorschrift erzeugt ein recht seltsames Bild (siehe Abbildung 4-4-3-1), dessen Interpretation auf den ersten Blick etwas schwer fällt. Überzeugen Sie sich selbst, indem Sie im folgenden Programm die Grenzen der Bildebene verändern.

Programm: *graf1.c*

```
/*********************************************************************/
/* graf1.c                                              20.01.88   */
/* Version 1.0                                                      */
/*                                                                  */
/*                                                                  */
/* fuer Markt+Technik geschrieben von Heiko Knappe                  */
/*********************************************************************/

#include <math.h>
#include <exec/types.h>
#include <intuition/intuition.h>

#define WIDTH  640        /* Screen Modus */
#define HEIGHT 500
#define DEPTH  2
#define FLAGS  HIRES+LACE

extern save();


struct IntuitionBase *IntuitionBase;
struct GfxBase *GfxBase;
struct Screen *MyScreen;
struct RastPort *rp;
struct NewScreen MyNewScreen=
 {0,0,WIDTH,HEIGHT,DEPTH,1,0,FLAGS,CUSTOMSCREEN,0,0,0,0};

short x,y,z;            /* absolute Punktkoordinaten */
float str,sti,r0=-20,i0=-20,r,i,rrange=80,irange=80;


/**********************/
/* Hauptprogramm      */
/**********************/

main(argc,argv)
int argc;
char *argv[];
{
 open_things();
 zeichnen();
 while(MyScreen->TopEdge<100);
 /* falls das Bild gespeichert werden soll */
 save("graf1.pic",1,MyScreen);
 close_things();
}


/**********************/
/* Initialisierungs-  */
/* routinen           */
/**********************/

open_things()
{
 if((IntuitionBase=OpenLibrary("intuition.library",0))==0)
    close_things();
 if((GfxBase=OpenLibrary("graphics.library",0))==0)
    close_things();
 if((MyScreen=OpenScreen(&MyNewScreen))==0)
    close_things();
 rp=&(MyScreen->RastPort);
 init_colors();
}

close_things()
{
 if(IntuitionBase!=0) CloseLibrary(IntuitionBase);
 if(GfxBase!=0) CloseLibrary(GfxBase);
 if(MyScreen!=0) CloseScreen(MyScreen);
 exit(0);
}
```

```
73
74  init_colors()
75  {
76   SetRGB4(&(MyScreen->ViewPort),Ø,Ø,Ø,Ø);
77   SetRGB4(&(MyScreen->ViewPort),1,15,Ø,15);
78   SetAPen(rp,1);
79  }
8Ø
81
82  /***********************/
83  /* Grafik zeichnen     */
84  /***********************/
85
86  zeichnen()
87  {
88   str=rrange/WIDTH,sti=irange/HEIGHT;
89   for(x=Ø;x<WIDTH;++x)
9Ø   {
91      r=rØ+x*str;
92      for(y=Ø;y<HEIGHT;++y)
93      {
94         if(MyScreen->TopEdge>1ØØ) return(Ø);
95         i=iØ+y*sti;
96         z=(short)(i*i+r*r);
97         SetAPen(rp,z%2);
98         WritePixel(rp,x,y);
99      }
1ØØ  }
1Ø1 }
```

Was zeigt uns nun dieses Bild? Wie wir wissen, beschreibt der Wert x^2+y^2 den Abstand vom Ursprung zum Quadrat. Dieser Wert ist also konstant für einen Kreis um (0,0). Man sagt, die Kreise um den Ursprung sind Äquipotentiallinien. Durch die Auswertung mit der Modulo-Funktion erwarten wir, daß abwechselnd schwarze und weiße Kreise gezeichnet werden. Leider erkennen wir im Bild diese Kreise um den Ursprung nicht. Im Gegenteil, es tauchen über das gesamte Bild viele kleine Kreise auf. Dies liegt daran, daß wir im Prinzip zwei Grundmuster miteinander verknüpft haben, ein Raster und ein Kreisringmuster.

Dabei wird an jedem Rasterpunkt dieses Ringmuster wiederholt. Jetzt ist es uns klar, wieso wir viele kleine Kreise sehen, und können die Übergänge zwischen den Zentren erklären. Im Prinzip interferieren die kleinen Kreismuster miteinander. Dieser Effekt, der hier ausgenützt wird, tritt oft auf und ist meist unerwünscht. Man denke hier nur an Optik und Abbildungsfehler durch Linsen.

Bei genauer Betrachtung leuchtet es aber ein, daß die Erwartung konzentrischer Kreise um den Ursprung falsch ist, denn angenommen, wir haben die beiden nebeneinanderliegenden Punkte (1Ø,1Ø) und (1Ø,11), so erhalten wir:

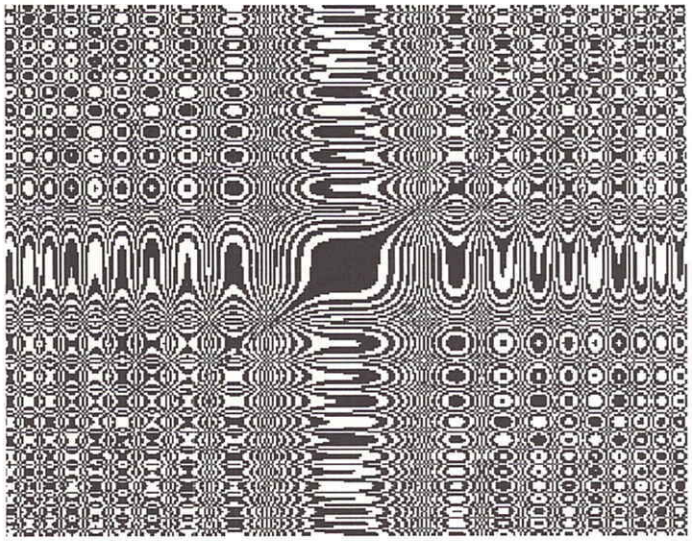
$(1Ø^2+1Ø^2) = 2ØØ \Rightarrow (2ØØ \text{ modulo } 2) = Ø$
$(1Ø^2+11^2) = 221 \Rightarrow (221 \text{ modulo } 2) = 1$

Daß keine Kreise um den Ursprung entstehen können, liegt an der großen Differenz von 21, die bereits zwischen benachbarten Punkten liegt. Es werden erst dann Kreise um den Ursprung entstehen, wenn wir aus x^2+y^2 die Wurzel ziehen, den Wert runden und dann die Modulo-Zuordnung vornehmen.

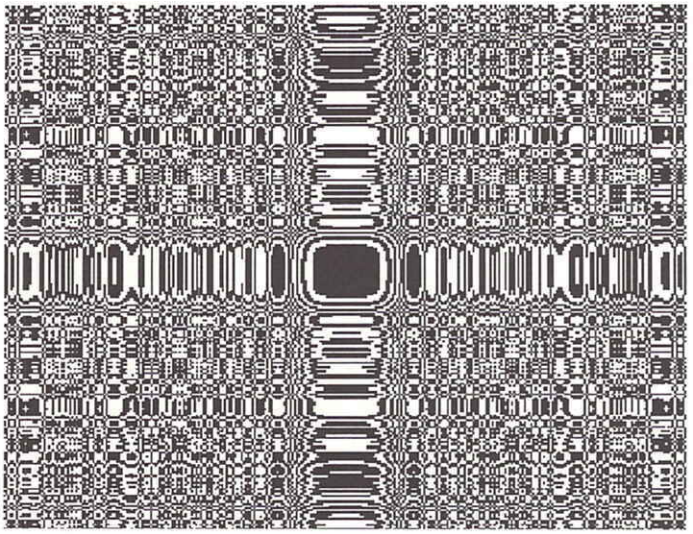
Wir können mit diesem Verfahren auch farbige Bilder erhalten. Dazu bewerten wir (x^2+y^2) mit modulo(Anzahl der Farben) und zeichnen den Punkt in der daraus erhaltenen Farbe. Im Programm *graf1.c* müssen wir das Screenformat, die Initialisierung der Farben ändern und die Berechnung entsprechend modifizieren:

```
#define WIDTH  32Ø          /* Screen Modus */
#define HEIGHT 25Ø
#define DEPTH  5
#define FLAGS  Ø

init_colors()
{
 int i;
 for(i=Ø;i<16;++i)
    SetRGB4(&(MyScreen->ViewPort),i,i,Ø,Ø);
 for(i=16;i<32;++i)
    SetRGB4(&(MyScreen->ViewPort),i,31-i,Ø,31-i);
}

zeichnen()
{
 str=rrange/WIDTH,sti=irange/HEIGHT;
 for(x=Ø;x<WIDTH;++x)
 {
    r=rØ+x*str;
    for(y=Ø;y<HEIGHT;++y)
    {
       if(MyScreen->TopEdge>1ØØ) return(Ø);
       i=iØ+y*sti;
       z=(short)(i*i+r*r);
       SetAPen(rp,z%32);
       WritePixel(rp,x,y);
    }
 }
}
```

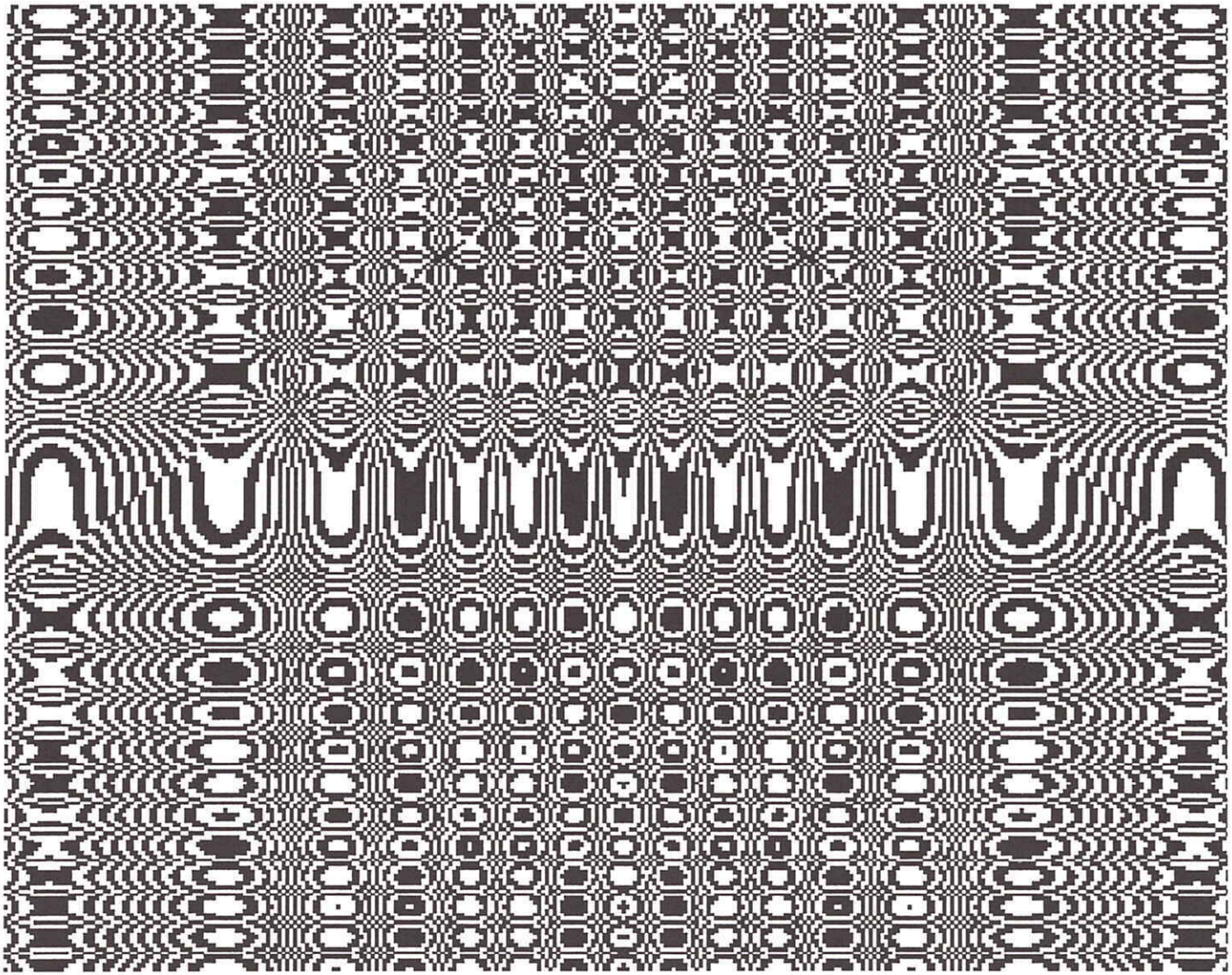
Nicht nur die Verwendung mehrerer Farben ist interessant! Warum sollten wir nicht eine andere Funktion zur Bewertung benutzen. Statt x^2+y^2 könnten wir doch zum Beispiel, x^3+y^3 (siehe Abbildung 4-4-3-2), x^4+y^4 (siehe Abbildung 4-4-3-3) oder `(cos(x)+sin(y))*1ØØØ` (siehe Abbildung 4-4-3-4) verwenden. Die Ergebnisse sind überraschend, oder?



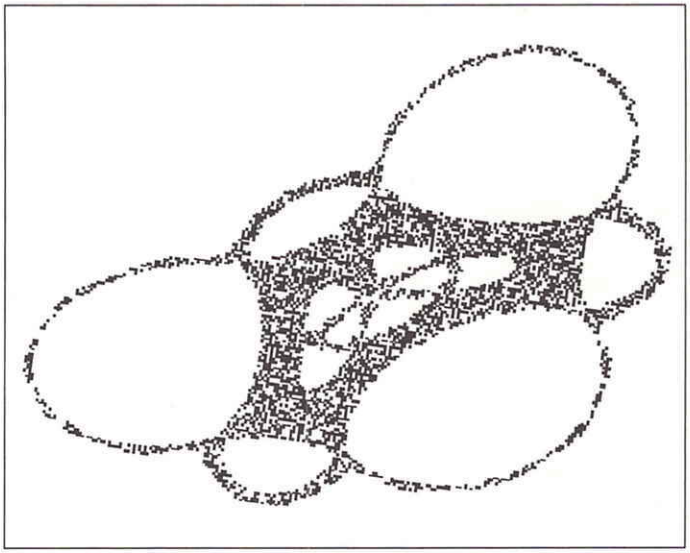
4-4-3-2 Funktion x^3+y^3



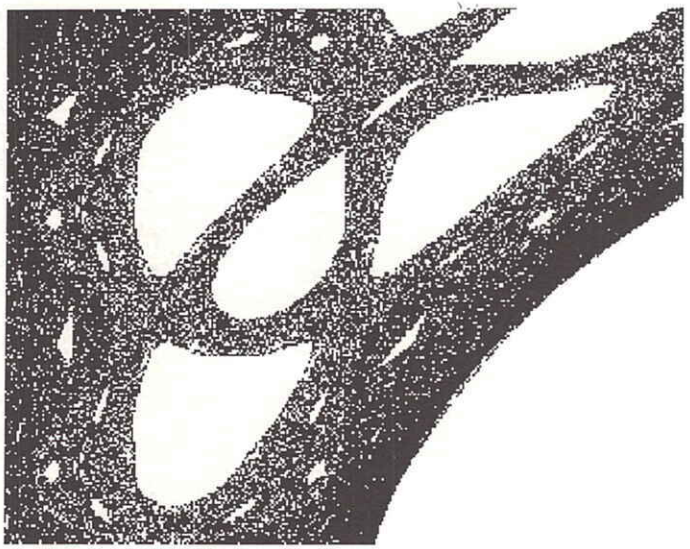
4-4-3-3 Funktion x^4+y^4



4-4-3-4 Funktion 1000(cos x + sin y)*



4-4-3-5



4-4-3-6

2) Einfache Rückkopplungen in der komplexen Zahlenebene

Das zweite Verfahren kennen wir bereits von dem Kapitel Feigenbäume. Der Berechnung legen wir, wie schon zu erwarten, eine rekursive Folge zugrunde. Wir starten mit dem Anfangswert x=y=Ø und zeichnen die Folgenglieder auf dem Bildschirm, sofern sie nicht außerhalb liegen. Wie die Formel aussieht, die wir verwenden wollen, bleibt uns selbst überlassen. Sie werden mit fast jeder ein recht schönes Bild erhalten. Ich möchte hier einige vorstellen:

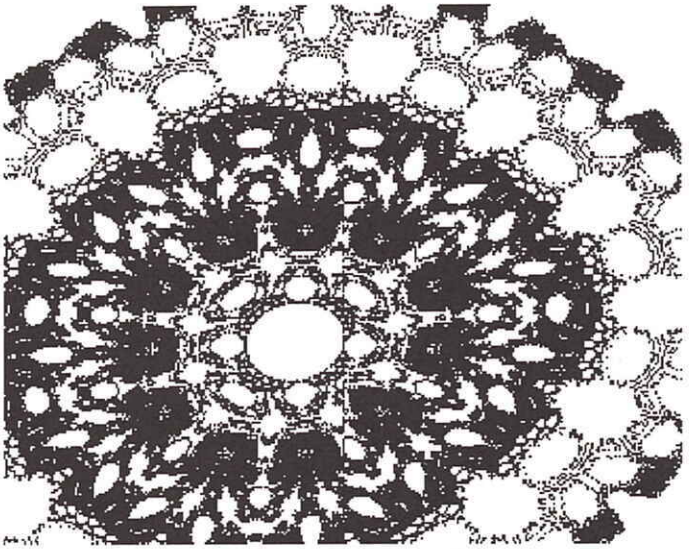
```
→ y = a-x                                          (1)
  x = y - sign(x)*√(abs(b*x - c))

→ y = a - x                                        (2)
  x = y - sin(x)

→ y = a - x                                        (3)
  x = y - √(x² - .2)
```

Bei der ersten Funktion müssen wir die drei Parameter a, b, c und bei den letzten beiden a fest vorwählen. Von ihnen wird sicher das Aussehen des fertigen Bildes bestimmt werden.

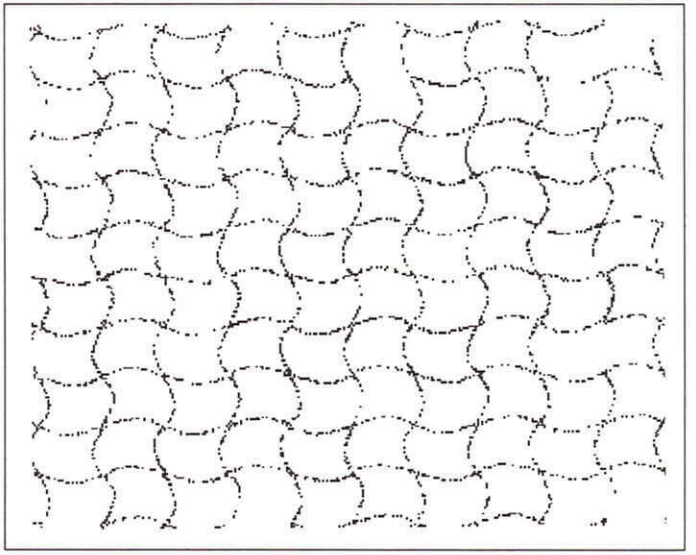
Sehen wir uns einmal die Ergebnisse an:

```
Funktion (1):
Abbildung 4-4-3-5: a=.4, b=1, c=Ø
(Bildebene [-2.5,2.5] [-2.5,2.5]
Abbildung 4-4-3-6: a=.4, b=1, c=Ø
(Vergrößerung aus Abbildung 4-4-3-5)
Abbildung 4-4-3-7: a=-3.14, b=c=.3
Funktion (2):
Abbildung 4-4-3-8: a=3.1
Funktion (3):
Abbildung 4-4-3-9: a=3.1
```

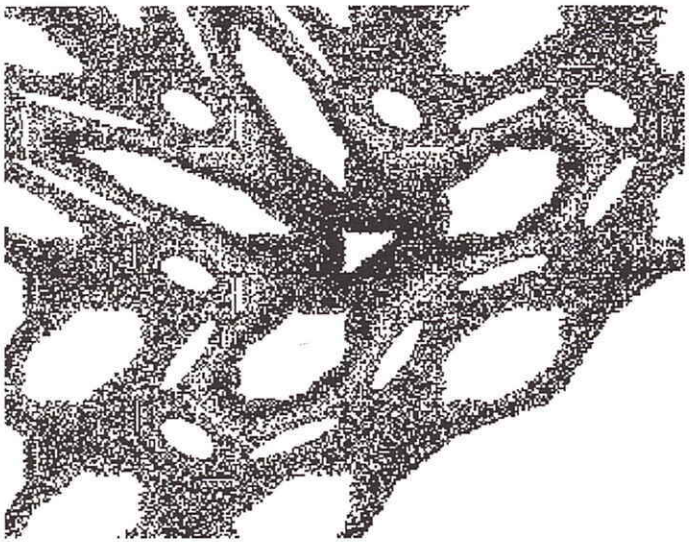
Bild 4-4-3-6 ist eine Vergrößerung von dem Zentrum des Bildes 4-4-3-5. Die filigranen Muster der Grafiken sind gerade in Vergrößerungen besonders gut zu erkennen. Wieder



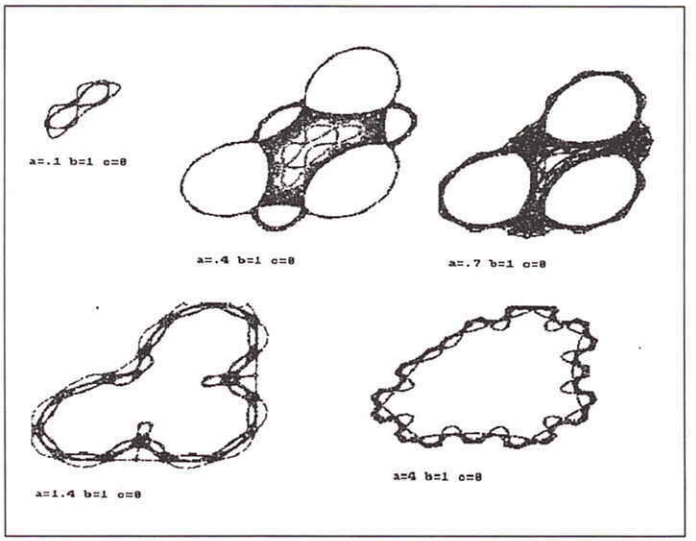
4-4-3-7 Mathematisches Gebilde, oder?



4-4-3-8 Immer wieder verblüffende Resultate

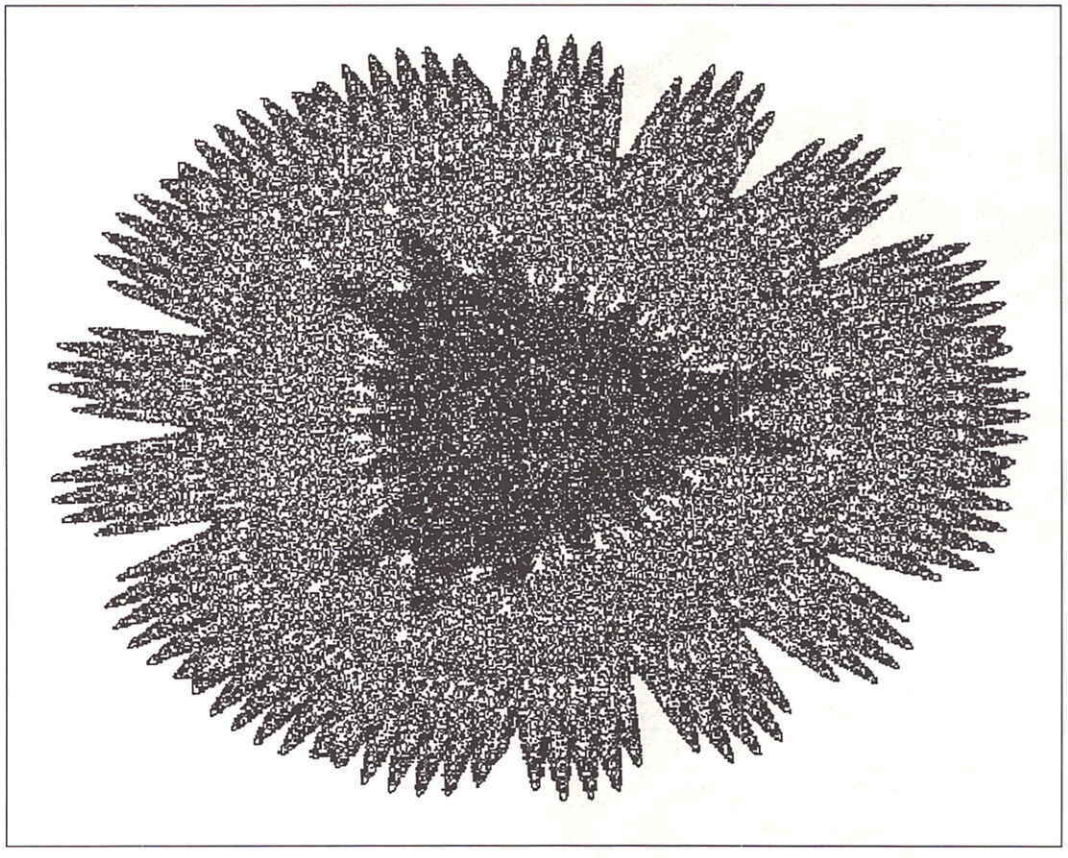


4-4-3-9 Filigranarbeit

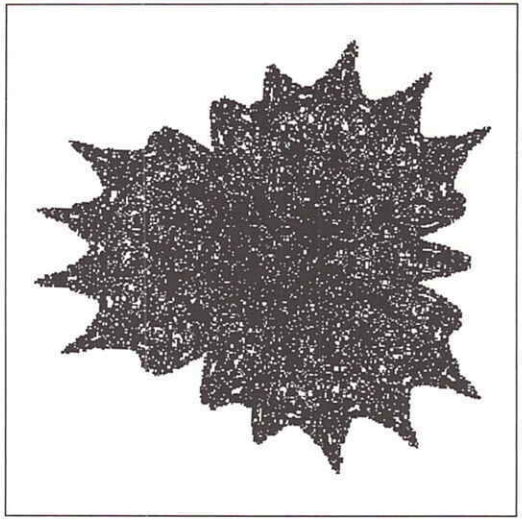


4-4-3-10 Selbst bei der Verwendung eines Rückkoppelungsprozesses entstehen unterschiedliche Bilder

4-4-3-11 (1) Weitere komplexe Rückkoppelungsprozesse

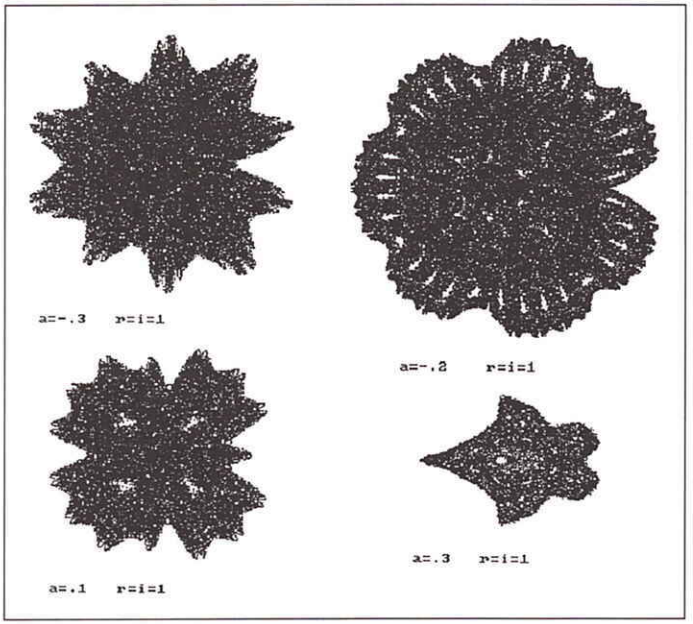


4-4-3-11 (2)

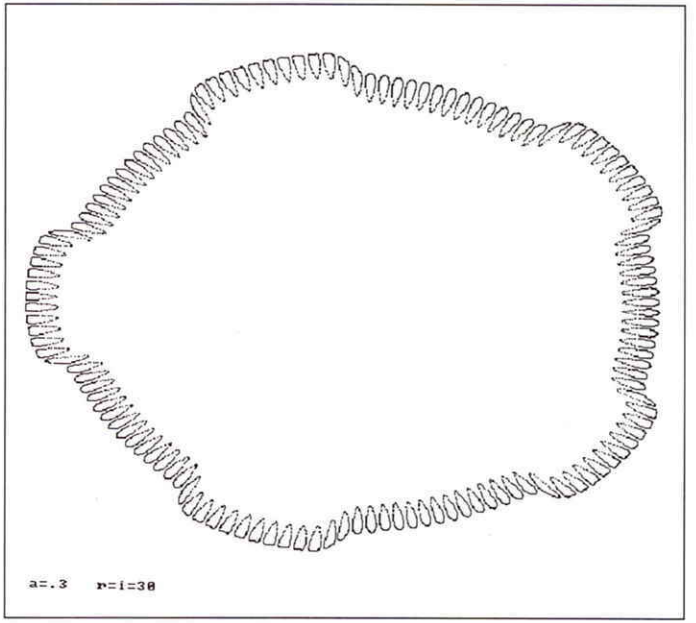


sehen wir die Eigenschaften der fraktalen Grafiken, die Selbstähnlichkeit und die unendliche Vielfalt der Strukturen. Kleine Veränderungen der Parameter ziehen ein vollständig neues Bild nach sich – Bild 4-4-3-7 zeigt dies. Verwenden wir nun aber auch noch andere Funktionen, so unterscheiden sich die Ergebnisse deutlich. (Siehe Zusammenstellung für einige Parameter zur Funktion (1) auch Abbildung 4-4-3-10.)

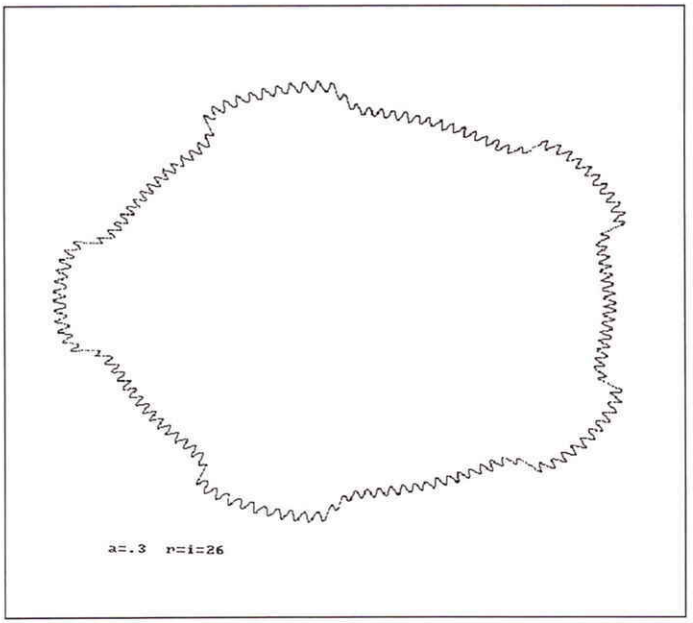
Interessant ist es auch, bei der Entstehung zuzuschauen. Scheinbar unregelmäßig werden Punkte im Bild gesetzt und formen nach und nach das endgültige Bild. Diese Art der Bildentstehung kennen wir bereits vom Henon-



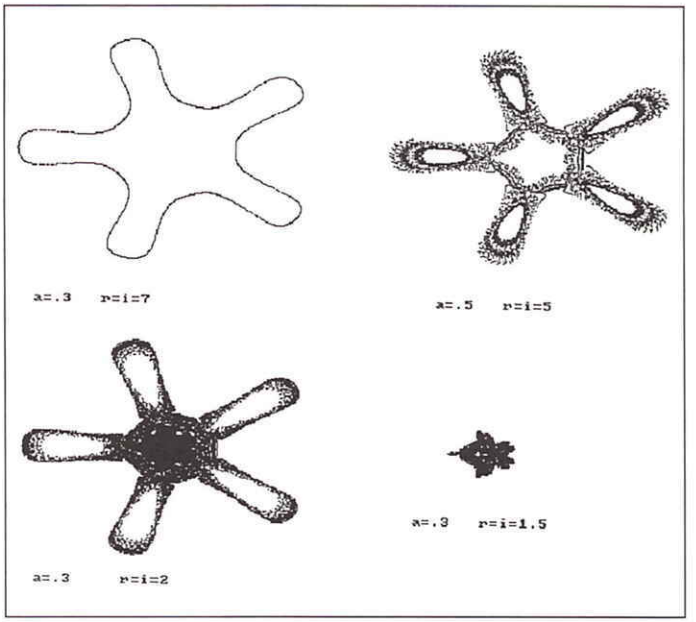
4-4-3-11 (3)



4-4-3-11 (4)



4-4-3-11 (5)



4-4-3-11 (6)

Attraktor und dem Programm *feigenbaum4.c.* Während es dort aber noch recht einfach war, das Aussehen zu verstehen, werden wir hier schon mehr Probleme haben, das fertige Bild vorherzusagen.

Noch eins zum Zeichnen der Punkte: Wir wollen, wenn möglich, wieder ein farbiges Bild generieren. Dies gelingt uns, indem wir den Punkt nicht mit einer bestimmten Farbe zeichnen, sondern die Farbregister der Farbe des Punktes mit `ReadPixel()` laden und den Punkt mit der Farbe des nächsthöheren Registers zeichnen. Die Realisierung im Programm wird also wie folgt aussehen:

```
SetAPen(MyRastPort, (ReadPixel
      (MyRastPort,x,y)+1)%32);

WritePixel(MyRastPort,x,y);
```

Damit ist auch sichergestellt, daß ein Punkt mit der Farbe des 31 Registers nun die Farbe des nullten Registers erhält.

Programm: *graf2.c*

```
/*********************************************************************/
/* graf2.c                                              20.01.88   */
/* Version 1.0                                                      */
/*                                                                  */
/*                                                                  */
/* x=y-sign(x)*sqrt(abs(b*x-c))                                     */
/* y=a-x                                                            */
/*                                                                  */
/* fuer Markt+Technik geschrieben von Heiko Knappe                  */
/*********************************************************************/

#include <math.h>
#include <exec/types.h>
#include <intuition/intuition.h>

#define WIDTH  320        /* Screen Modus */
#define HEIGHT 250
#define DEPTH  5
#define FLAGS  0

#define XOFF   160
#define YOFF   125

extern save();


struct IntuitionBase *IntuitionBase;
struct GfxBase *GfxBase;
struct Screen *MyScreen;
struct RastPort *rp;
struct NewScreen MyNewScreen=
 {0,0,WIDTH,HEIGHT,DEPTH,1,0,FLAGS,CUSTOMSCREEN,0,0,0,0};

float i=0,r=0,ii,rr;     /* relative Punktkoordinaten */
float a=.4,b=1,c=0;      /* Parameter */
float str,sti,rrange=2.5,irange=2.5;
short x,y;



/**********************/
/* Hauptprogramm      */
/**********************/

main(argc,argv)
int argc;
char *argv[];
{
 open_things();
 zeichnen();
 while(MyScreen->TopEdge<100);
 /* falls das Bild gespeichert werden soll */
 save("graf21.pic",1,MyScreen);
 close_things();
}
```

```
58 /************************/
59 /* Initialisierungs-   */
60 /* routinen            */
61 /************************/
62
63 open_things()
64 {
65  if((IntuitionBase=OpenLibrary("intuition.library",Ø))==Ø)
66     close_things();
67  if((GfxBase=OpenLibrary("graphics.library",Ø))==Ø)
68     close_things();
69  if((MyScreen=OpenScreen(&MyNewScreen))==Ø)
70     close_things();
71  rp=&(MyScreen->RastPort);
72  init_colors();
73 }
74
75 close_things()
76 {
77  if(IntuitionBase!=Ø) CloseLibrary(IntuitionBase);
78  if(GfxBase!=Ø) CloseLibrary(GfxBase);
79  if(MyScreen!=Ø) CloseScreen(MyScreen);
80  exit(Ø);
81 }
82
83 init_colors()
84 {
85  int i;
86
87  for(i=Ø;i<16;++i)
88     SetRGB4(&(MyScreen->ViewPort),i,i,Ø,Ø);
89  for(i=16;i<32;++i)
90     SetRGB4(&(MyScreen->ViewPort),i,31-i,Ø,31-i);
91 }
92
93
94 /************************/
95 /* Grafik zeichnen     */
96 /************************/
97
98 zeichnen()
99 {
100  str=(float)(WIDTH-XOFF)/rrange,sti=(float)(HEIGHT-YOFF)/irange;
101  while(MyScreen->TopEdge<1ØØ)
102  {
103     x=XOFF+(short)(r*str),y=YOFF-(short)(i*sti);
104     while((x<Ø)(x>WIDTH)(y<Ø)(y>HEIGHT))
105     {
106        if(x<Ø) x=-x;
107        if(x>WIDTH) x=2*WIDTH-x;
108        if(y<Ø) y=-y;
109        if(y>HEIGHT) y=2*HEIGHT-y;
110     }
111     SetAPen(rp,(ReadPixel(rp,x,y)+1)%32);
112     WritePixel(rp,x,y);
113     rr=(b*r-c); if(rr<Ø) rr=-rr;
114     rr=sqrt(rr); if(r<Ø) rr=-rr;
115     rr=i-rr;
116     ii=a-r;
117     r=rr,i=ii;
118  }
119 }
```

Im obigen Programm wurden auch die Teile der Kurve gezeichnet, die eigentlich nicht in der Bildebene liegen, in dem sie an den Bildbegrenzungen gespiegelt wurden. Ist dies nicht erwünscht, so sind im Programm die Zeilen 104 bis 112 durch

```
if((x>=Ø)&&(x<WIDTH)&&(y>=Ø)&&(y<HEIGHT))
 {
    SetAPen(rp,(ReadPixel(rp,x,y)+1)%32);
    WritePixel(rp,x,y);
 }
```

zu ersetzen.

Viel Spaß beim Experimentieren!

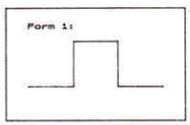
4.4.4 Einige Anmerkungen zu den vorgestellten Fraktalen

Im Prinzip wird bei diesen immer so vorgegangen, daß ein Streckenzug durch eine Folge neuer ersetzt wird – das Vorgehen ist also sehr einfach. Versucht man es aber in einem Programm zu realisieren, so werden einige Probleme auftauchen. Wir müssen an dieser Stelle etwas umdenken, wie wir an den ersten Beispielen gesehen haben. Es gelingt uns, das Zeichnen auf einen rekursiven Prozeß zurückzuführen, indem wir die Schildkrötengrafik aus LOGO verwenden. LOGO wurde in den sechziger Jahren von Seymor Papert am Massachusetts Institute of Technology (MIT) entwickelt. Mit LOGO entfernen wir uns vom großen Überblick und begeben uns auf eine tiefere Ebene. Wir versuchen uns vorzustellen, wie wir vorgehen würden, wenn wir selbst die gewünschte Grafik zeichnen wollten. Zum Beispiel wird ein Kreis nicht mehr durch einen Mittelpunkt und einen Radius charakterisiert, sondern wird in kleine Teilschritte zerlegt. So gehen wir zum Zeichnen eines Kreises mit der Schildkröte immer ein kleines Stück vorwärts und drehen uns dann um einen kleinen Winkel bis wir 360° durchlaufen haben. Der Radius des Kreises wird durch die Schrittweite beim Vorwärtsgehen und dem Winkel bei der Drehung bestimmt. Es ist ganz klar, daß dies eine völlig andere Darstellungsweise eines Kreises ist. Das ist der Reiz der Sprache LOGO, die zum Nachdenken über die praktische Realisierung auffordert.

Sie eignet sich geradezu zum Zeichnen fraktaler Grafik der Dimension zwischen eins und zwei, denn Bewegungen der Schildkröte, sei es ein Drehen oder ein Vorwärtsgehen, hängt immer von ihrer augenblicklichen Position und Orientierung ab. Ihr Zustand beinhaltet sozusagen die notwendige Rückkopplung. Man bezeichnet die Geometrie der Schildkröte als formal endliche Differentialgeometrie, denn sie kann sich nur in diskreten Schritten bewegen und die Angabe der Schrittweite ist nichts anderes als die Differenz zwischen der alten und der neuen Position. Dies bedeutet nichts anderes, als daß sich diese Art der Geometrie allein um die lokalen Eigenschaften kümmert und auf die globalen Eigenschaften einer Grafik keinen Bezug nimmt.

Als Ergänzungen zu den Funktionen `Forward()`, `Backward()` und `Turn()` ist es ganz sinnvoll noch zwei weitere Funktionen einzuführen: `Penup()` und `Pendown()`, um die Schildkröte auch ungesehen über den Schirm bewegen zu können.

Wollen wir zum Beispiel eine Gerade (Länge `1`) durch einen Streckenzug der Form `1` ersetzen, so würde ein Programm wie folgt aussehen:




```
forward(1/3); turn(-90); forward(1/3); turn(90);
forward(1/3); turn(90); forward(1/3); turn(-90)
forward(1/3);
```

Die nächste Unterteilung würde dann so aussehen:

Seite 1

```
forward(1/6); turn(-90); forward(1/6); turn(90);
forward(1/6); turn(90); forward(1/6); turn(-90);
forward(1/6);
```

Seite 2

```
forward(1/6); turn(-90); forward(1/6); turn(90);
forward(1/6); turn(90); forward(1/6); turn(-90);
forward(1/6);
```

Seite 3

```
forward(1/6); turn(-9Ø); forward(1/6); turn(9Ø);
forward(1/6); turn(9Ø); forward(1/6); turn(-9Ø);
forward(1/6);
```

Seite 4

```
forward(1/6); turn(-9Ø); forward(1/6); turn(9Ø);
forward(1/6); turn(9Ø); forward(1/6); turn(-9Ø);
forward(1/6);
```

Seite 5

```
forward(1/6); turn(-9Ø); forward(1/6); turn(9Ø);
forward(1/6); turn(9Ø); forward(1/6); turn(9Ø);
forward(1/6);
```

Wir erkennen leicht die Ordnung, die zu erwarten war. Nach einigem Überlegen können wir allgemein den Algorithmus wie folgt formulieren:

```
fraktal(seite,winkel)
short seite,winkel;
{
 if(winkel!=Ø)
 {
    fraktal(seite/3,winkel-1); turn(-9Ø);
    fraktal(seite/3,winkel-1); turn(9Ø);
    fraktal(seite/3,winkel-1); turn(-9Ø);
    fraktal(seite/3,winkel-1); turn(9Ø);
    fraktal(seite/3,winkel-1);
 }
 else forward(seite);
}
```

Die Variable `winkel` gibt an, wie oft die Kurve unterteilt wird, das heißt, wie groß die Anzahl der vorgenommenen Iterationen ist. Analog läßt sich jede andere Streckenzugunterteilung realisieren. Jetzt bleibt mir nur noch, viel Spaß beim Experimentieren zu wünschen.

Daß sich für Mathematiker die Untersuchung von Fraktalen nicht aufs bloße Experimentieren und die Suche nach schönen Bildern beschränkt, zeigt folgende Aussage:

Die Koch-Kurve hat eine Dimension von 1.2618!

Sicher wird es jeden, der dies zum erstenmal liest, etwas seltsam anmuten, von einer gebrochenen Dimension zwischen eins und zwei zu sprechen. Diese Dimension darf nicht sofort als Unsinn abgetan werden (habe ich zu Beginn auch getan), sondern man muß sie als eine rein mathematische Größe ansehen. Sie hat nichts mit der unmittelbaren Vorstellung einer anschaulichen Dimension (D_T topologische Dimension) zu tun – es ist eine fraktale Dimension (D)! Sie ist geeignet eine fraktale Kurve zu beschreiben. Ist ihr Wert zum Beispiel nahe bei eins, so handelt es sich um ein Fraktal, der nur wenig strukturiert und einer Geraden ähnlich ist. Strebt die Dimension dagegen dem Wert zwei zu, so bedeutet dies, daß die fraktale Kurve sehr verschnörkelt und flächenfüllend ist.

Mit Hilfe der fraktalen Dimension D ist es auch möglich zu definieren, was ein Fraktal eigentlich ist:

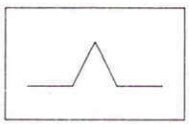
Ein Fraktal ist nach Definition eine Menge, deren Hausdorff-Besicovitch-Dimension D echt die topologische Dimension D_T übersteigt.

Das heißt also: Eine Menge, deren topologische Dimension D_T kleiner als D ist, ist ein Fraktal. Also vor allem: Ist D nicht ganzzahlig, haben wir ganz sicher ein Fraktal vor uns. Mandelbrot (in Die fraktale Geometrie der Natur) begründet die Richtigkeit dieser Aussage folgendermaßen: Stellen wir uns vor, wir hätten die Aufgabe, die Länge einer Küstenlinie zu bestimmen. Es wird jedem einsichtig sein, daß wir stark schwankende Ergebnisse

erhalten, je nachdem welchen Meßmaßstab wir anlegen werden. Nehmen wir zum Beispiel einen Meterstab und messen mit ihm die Küste aus, so werden wir sicher ein kleineres Ergebnis erhalten, als wenn wir in Zentimetern messen würden. Je kleiner wir die Vergleichsgröße 1 machen, desto größer wird die gemessene Gesamtlänge L. Es gilt: $L(1) = N * 1$, wobei N die Anzahl der Schritte bedeutet. Mit dieser Meßmethode gelingt es uns also nicht, eine vernünftige Messung vorzunehmen. Nach Richardson (1961) gibt es zwei Konstanten, α und D, nach denen es genügt die Küstenlinie mit $\alpha * 1^{-D}$ Intervallen als Polygonzug zu approximieren, um ein vernünftiges Ergebnis zu erhalten. Für die Gesamtlänge gilt also:

$$L(1) \text{ ca. } \alpha * 1^{-D} * 1 = \alpha * 1^{1-D}$$

Nun können wir auch verifizieren, warum die Koch-Kurve eine Dimension von 1.2618 hat. Bei der Koch Kurve wird jede Strecke durch folgenden Streckenzug ersetzt:



Offenbar gilt $L(1) = 1$ und $L\left(\frac{1}{3}\right) = \frac{4}{3} * L(1)$

und wenn wir diese beiden Beziehungen in $L(1)=1^{1-D}$ einsetzen:

$$\left(\frac{1}{3}\right)^{1-D} = \frac{4}{3} * 1^{1-D}$$

$$\frac{1^{1-D}}{3^{1-D}} = \frac{4}{3} * 1^{1-D}$$

$$3^{D-1} = \frac{4}{3}$$

$$3^{D} = 4$$

$$D = \frac{\log(4)}{\log(3)} = 1.2618$$

Eine andere Argumentation fördert die Verständlichkeit der Existenz einer gebrochen rationalen Dimension ebenfalls, so daß ich sie hier ebenfalls vorstellen möchte. Also wer bisher noch etwas ungläubig diese fraktalen Dimensionen betrachtet, der versteht vielleicht dieses Vorgehen:

Der Mathematiker Felix Haussdorff hat einen Dimensionsbegriff geprägt, den der Hausdorff-(Besicovitch-)Dimension.

Wie bei der Messung einer Küstenlinie wollen wir versuchen, einfache geometrische Mengen auszumessen – ein Quadrat der Fläche 1 und eine Linie der Länge 1. Dazu verwenden wir kleine Quadrate der Kantenlänge α; sie haben also den Flächeninhalt α^2. Für die Messung der Linie werden wir $\frac{1}{\alpha}$ Meßquadrate aneinanderlegen müssen, für die Fläche $\frac{1}{\alpha^2}$.

$$F_{Quadrat} = \frac{1}{\alpha^2} * \alpha^2 = 1$$

Dieses Ergebnis erstaunt uns nicht weiter. Legen wir aber auch bei der Messung der Geradenlänge das Meßquadrat α^2 zugrunde, so erhalten wir:

$$F_{Gerade} = \frac{1}{\alpha} * \alpha^2$$

Dieser Wert geht für kleine α gegen Null. Auch dies ist eigentlich nicht weiter seltsam, denn eine Gerade hat eben nun einmal keine Breite und damit auch keine Fläche. Wollen wir die Länge der Geraden messen, so müssen wir die Meßeinheit gleich α setzen. Nun erhalten wir:

$$L_{Gerade} = \frac{1}{\alpha} * \alpha = 1$$

Verwenden wir diese Einheit aber für die Messung der Quadratfläche, so werden wir umgekehrt auch hier kein sinnvolles Ergebnis

erwarten können, denn das Ergebnis würde für kleine α gegen unendlich gehen.

Allgemein kann man nun sagen, daß nur für eine bestimmte Größe der Meßeinheit α^D, also für ein bestimmtes D, der Wert der Messung einen endlichen Wert annimmt. Wählen wir D größer, so ist das Ergebnis gleich Null, wählen wir D kleiner, so erhalten wir unendlich. Dieses D nennt man nun Dimension der Menge. Für eine Fläche ist die Dimension gleich 2, für eine Gerade gleich 1 – dies war zu erwarten.

Betrachten wir nun aber ein fraktales Gebilde wie die Koch-Kurve: Für die Länge der fraktalen »Küstenlinie« erhalten wir:

$$L\left(\frac{1}{3}\right) = \frac{4}{3} * L(1)$$

Da für L

$$L(1) = 1^{-D} * 1 = 1^{1-D}$$

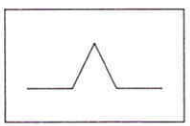
gelten soll, erhalten wir bereits wie oben das Ergebnis:

$$D = 1.2618$$

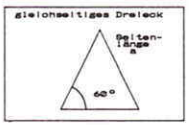
Ich hoffe, es ist mir gelungen, den Begriff der fraktalen Dimension so zu erklären, daß er verständlich geworden ist und nicht mehr als Unsinn abgetan wird.

4.4.5 Das seltsame Verhalten fraktaler Kurven – oder Umfang und Inhalt einer Kochschen Schneeflocke

Wir kennen bereits die Koch-Kurve, die durch Ersetzen einer Seite durch



entsteht. Beginnen wir nun als Ausgangsfigur nicht wie bisher mit einer Geraden, sondern mit einem Dreieck, so entsteht nach mehreren Durchgängen ein Gebilde, das einer Schneeflocke ähnlich sieht. Deshalb wird es auch die Kochsche Schneeflocke genannt. Wir wollen jetzt die Länge der Umfangslinie und die eingeschlossene Fläche nach dem n-ten Durchgang berechnen. Dazu müssen wir uns zuerst einmal überlegen, welchen Inhalt $A_\emptyset$, ein gleichseitiges Dreieck mit der Seitenlänge a besitzt:



$$A_\emptyset = \frac{a}{2} * a * \sin(60°) = \frac{a^2}{4} * \sqrt{3}$$

Beginnen wir mit einem gleichseitigen Dreieck der Seitenlänge a, so ist diese Fläche $A_\emptyset$ die Startfläche. Beim nächsten Durchgang erhalten wir für jede Seite ein neues Dreieck mit dem Inhalt A_1':

$$A_1' = \frac{1}{4} * \left(\frac{a}{3}\right)^2 * \sqrt{3}$$

Die neue Gesamtfläche erhalten wir also zu:

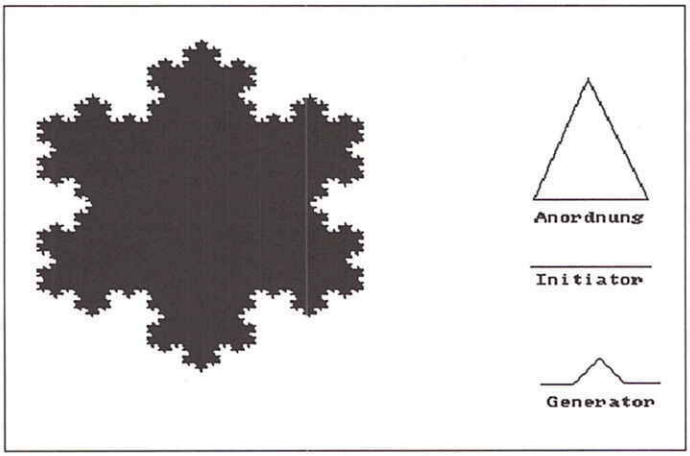
$$A_1 = A_\emptyset + 3*A_1' = A_\emptyset * \left(1 + \frac{1}{3}\right)$$

Um allgemein die eingeschlossene Fläche der Figur anzugeben, müssen wir verstehen, wie viele Dreiecke beim n-ten Durchlauf neu entstehen. Jede Seite hat n-1-Kanten, die bei diesem Durchgang neu unterteilt werden. Berücksichtigen wir außerdem, daß die Grundfigur drei Seiten hatte, so erhalten wir

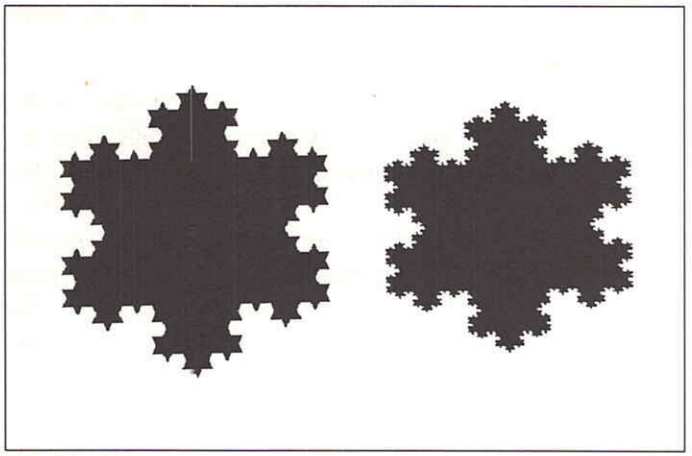
$$3 * 4^{n-1}$$

neue Dreiecke. Jedes dieser Dreiecke schließt eine Fläche A_n' von

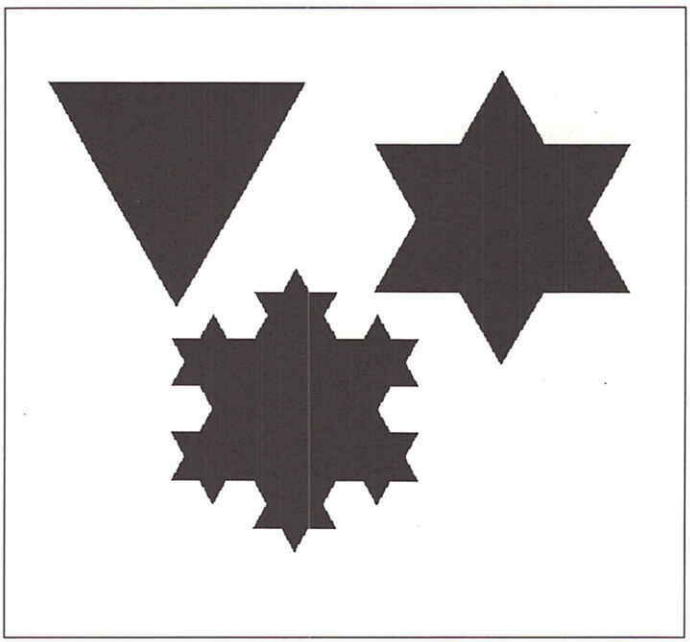
$$A_n' = \frac{1}{4} * \left(\frac{a}{3^n}\right)^2 * \sqrt{3}$$



4-4-5-1 Die Kochsche Schneeflocke



4-4-5-2 (2)



4-4-5-2 (1) Verschiedene Stadien

ein. Damit erhalten wir die Fläche A_n des n-ten Durchlaufs in Abhängigkeit von A_{n-1} zu:

$$A_n = A_{n-1} + 3 * 4^{n-1} * \frac{(a/3^n)^2}{4} * \sqrt{3} =$$

$$= A_{n-1} + \frac{a^2}{4} * \sqrt{3} * \frac{4^{n-1}}{3^{2n-1}}$$

Unter Verwendung der Fläche $A_\emptyset$ erhalten wir:

$$A_n = A_\emptyset * \left(1 + \frac{1}{3} + \ldots + \frac{4^{n-1}}{3^{2n-1}}\right)$$

Für den Umfang der Figur U_n nach dem n-ten Durchlauf erhalten wir:

$$U_n = \frac{a}{3^n} * 3 * 4^n = a * 3 * \left(\frac{4}{3}\right)^n$$

Dabei ist $\frac{a}{3^n}$ die Länge der einzelnen Linienstücke und $3*4^n$ die Anzahl der Stücke nach der n-ten Iteration.
Betrachten wir nun die Figur nach »Unendlich« vielen Schritten, so erhalten wir für den Umfang U

$$U = \lim_{n \to \infty} \left(a * 3 * \left(\frac{4}{3}\right)^n\right) \rightarrow \text{Unendlich},$$

da $\frac{4}{3}$ größer als eins ist. Der Umfang wächst also ständig an. Dagegen strebt die eingeschlossene Fläche A gegen einen festen Grenzwert, denn die Summe

$$\left(1 + \frac{1}{3} + \ldots + \frac{4^{n-1}}{3^{2n-1}}\right)$$

kann so umgeformt werden, daß sie als eine geometrische Reihe zu erkennen ist:

$$1 + \frac{1}{3} \sum_{k=0}^{n} \left(\frac{4}{9}\right)^k = 1 + \frac{1}{3} * \frac{1-\left(\frac{4}{9}\right)^{n+1}}{1-\frac{4}{9}}$$

Diese Reihe strebt für n gegen Unendlich gegen den Wert $\frac{8}{5}$.

Für die Fläche A erhalten wir also den Grenzwert

$$A = \frac{8}{5} A_{\emptyset}.$$

Der Umfang der Figur wächst demnach mit jedem Schritt unbegrenzt an, wohingegen die eingeschlossene Fläche gegen einen festen Grenzwert strebt. Dieses Ergebnis ist doch ein wenig unerwartet, oder?

5 Wie die Bilder laufen lernten – oder ein Flug ins Apfelmännchen

Jetzt sind wir so ziemlich am Ende des Buches angekommen, haben Apfelmännchen, Julia-Mengen und andere kuriosen Mengen kennengelernt. Wir haben dreidimensionale Gebirge für die zugrundeliegenden Folgen berechnet und haben uns langsam mit den Mengen vertraut gemacht. Jetzt finden wir uns schon ganz gut in ihnen zurecht und können die Bilder interpretieren. Jede Auswertung hat uns dem Ziel, dem Verständnis der dynamischen Systeme, nähergebracht. Obwohl die Bilder sehr anschaulich sind, bleiben sie doch etwas Statisches. Wie gerne würden wir die Auswirkungen einer Veränderung der komplexen Konstanten c bei den Julia-Mengen kontinuierlich sehen können, um Bilder unmittelbar zu vergleichen. Oder wie wäre es mit einem Flug durch eine dreidimensionale Apfelmännchen-Landschaft?
Nun ja, zuviel erwarten darf man nicht. Die Berechnung der Bilder dauert zu lange, als daß aus ihnen ein Echtzeitfilm entstehen könnte, dessen Abspielgeschwindigkeit uns befriedigt. Also bleibt nur eins übrig: Bilder zuerst berechnen und sie dann abspielen. Dabei ist es nicht möglich, die Bilder immer von Diskette zu laden, denn auch dies wäre zu langsam. Um einen durchgehenden Film zu erhalten, müssen die Grafiken zuerst geladen und im Speicher abgelegt werden. Damit ist aber durch den begrenzten Speicher die Anzahl der Bilder und somit die Länge der Sequenz stark eingeschränkt, die Abspielgeschwindigkeit jedoch befriedigend.

Das Abspielen der Bilder erfolgt folgendermaßen: Nachdem die Grafiken geladen und im Speicher in eigens eingerichteten BitPlanes abgelegt wurden, werden die einzelnen Bilder durch `LoadView()` angezeigt. Jedes Bild erhält eine eigene View-Struktur etc. ist also ein völlig eigenständiges Bild. Deshalb ist die Verwendung von `LoadView()` auch möglich. Wir verwenden bei diesem Vorgehen keine Intuition-Funktionen, sondern machen »alles« selbst. Mit dieser Methode begrenzen wir den nutzbaren Speicher auf das CHIP-Memory, auf den der Blitter zugreifen kann, denn dieser bewerkstelligt das blitzschnelle Einblenden der Bilder bei Aufruf von `LoadView()`. Deshalb ist die Anzahl der ladbaren Bilder bei einem Format von 320 x 200 x 5 auf zehn Bilder begrenzt. Im Prinzip wäre es möglich, den weiteren Speicher (FAST-Memory) zu nutzen und die Daten von dort nach unten in den CHIP-Mem zu kopieren und damit mehr Bilder abspielen zu können. Dies Verfahren wird von einem Public-Domain-Programm mit dem Namen *LMV* angewendet. Bei 2.5 Mbyte Speicher werden dann bis zu ca. 30 Bildern möglich. Da die Verwaltung der Bilddaten aber wesentlich aufwendiger werden und der Umfang des Programms anwachsen würde, habe ich mich auf die Verwendung des CHIP-Mem beschränkt.

Für diejenigen, die sich genauer mit dem Projektor auseinandersetzen wollen, ist es

zuerst wichtig zu wissen, wie ein Screen ohne Verwendung des Intuition geöffnet wird. In folgender Reihenfolge müssen wir dazu folgende Strukturen einrichten:

- *View*-Struktur
- *ViewPort*-Struktur
- *RasInfo*-Struktur
- *BitMap*-Struktur
- *RastPort*-Struktur

Wie man bei der Initialisierung vorgeht zeigt das folgende Programm *screen.c*.

Programm: *screen.c*

```
#include <exec/types.h>
#include <exec/exec.h>
#include <graphics/gfx.h>
#include <graphics/gfxmacros.h>
#include <graphics/copper.h>
#include <graphics/gels.h>
#include <graphics/regions.h>
#include <graphics/clip.h>
#include <graphics/text.h>
#include <graphics/gfxbase.h>
#include <graphics/view.h>
#include <hardware/dmabits.h>
#include <hardware/custom.h>
#include <hardware/blit.h>

#define DEPTH   5
#define WIDTH   320
#define HEIGHT  250

struct View MyView;
struct View *OldView;
struct ViewPort MyViewPort;
struct RasInfo MyRasInfo;
struct BitMap MyBitMap;
struct RastPort MyRastPort;
struct GfxBase *GfxBase;

USHORT i;


main()
{
 init_all();
 Delay(50*10); /* Warten */
 close_all();
}

init_all()
{
 if((GfxBase=OpenLibrary("graphics.library",0))==0)
    exit(FALSE);
 OldView=GfxBase->ActiView; /* alten View retten */
 InitView(&MyView); /* View-Struktur initialisieren */
 InitVPort(&MyViewPort); /* ViewPort-Struktur initialisieren */
 MyView.ViewPort=&MyViewPort;
 MyViewPort.DWidth=WIDTH;
 MyViewPort.DHeight=HEIGHT;
 MyViewPort.RasInfo=&MyRasInfo;
 InitBitMap(&MyBitMap,DEPTH,WIDTH,HEIGHT); /* BitMap initialisieren */
 MyRasInfo.BitMap=&MyBitMap; /* RasInfo-Struktur initialisieren */
 MyRasInfo.RxOffset=0;
 MyRasInfo.RyOffset=0;
 MyRasInfo.Next=0;
 /* BitMap-Speicher allocatieren */
 for(i=0;i<DEPTH;++i)
    if((MyBitMap.Planes[i]=AllocRaster(WIDTH,HEIGHT))==0)
       exit(FALSE);
 MakeVPort(&MyView,&MyViewPort);
 MrgCop(&MyView);
 for(i=0;i<DEPTH;++i)
    BltClear(MyBitMap.Planes[i],RASSIZE(WIDTH,HEIGHT),0);
 LoadView(&MyView);
 /* Farbe initialisieren */
 SetRGB4(&MyViewPort,0,15,0,0);  /* rot */
}


close_all()
{
 LoadView(OldView); /* alte View-Struktur wieder initialisieren */
```

```
71  for(i=Ø;i<DEPTH;++i)
72     FreeRaster(MyBitMap.Planes[i],WIDTH,HEIGHT);
73  FreeVPortCopLists(&MyViewPort);
74  FreeCprList(MyView.LOFCprList);
75  CloseLibrary(GfxBase);
76 }
```

Da die Laderoutinen des IFF-Moduls nicht geeignet sind, in eigene BitPlanes zu schreiben, war es nötig, sie für den Projektor zu modifizieren. Beim Starten des Projektors geht man folgendermaßen vor:

```
pro.run Bild1 Bild2 ...
```

Das Programm lädt dann automatisch die Bilder in der angegebenen Reihenfolge. Tritt ein Fehler auf, so teilt das Programm dies mit. Ist ein Bild-File nicht ladbar, so wird es übersprungen und versucht, die weiteren Bilder zu laden. Ist nicht mehr genug Speicher frei, so wird das Laden abgebrochen. (Fehler 1: das Bild-File konnte nicht geladen werden / Fehler 2: das File ist kein IFF-Format / Fehler 100: nicht genug Speicher.)

Sind die Bilder geladen, wird mitgeteilt wieviele geladen werden konnten und es wird ein Consol-Fenster geöffnet über das alle weiteren Eingaben erfolgen. Folgende Eingaben sind möglich:

`e Ø oder 1`	Einzelbild an (1) oder aus (0)
`p 1/Ø , 1/Ø`	Vorwärts an/aus, Rückwärts an/aus
`n anz`	Anzahl der Durchläufe
`z zeit`	Abspielrate
`b start,ende`	Anfangs-, Endbild beim Abspielen
`a`	Ausgabe der gesetzten Parameter
`d`	Default-Werte setzen
`s`	Film starten
`x`	Programm verlassen

Programm: *pro.c*

```
 1 /*************************************************************************/
 2 /* pro.c                                                  27.Ø1.88  */
 3 /* Version 2.Ø                                                      */
 4 /*                                                                  */
 5 /*                                                                  */
 6 /* Projektor zum Abspielen mehrerer IFF-Bilder hintereinander       */
 7 /*                                                                  */
 8 /* fuer Markt+Technik geschrieben von Heiko Knappe                  */
 9 /*                                                                  */
1Ø /* Bilder im IFF-Format koennen geladen werden und dann schnell     */
11 /* hintereinander abgespielt werden. Dadurch entsteht die Wirkung   */
12 /* einer kurzen Filmsequenz. Wieviele Bilder geladen werden koennen */
13 /* haengt vom verfuegbaren Speicher ab                              */
14 /* Der Aufruf des Programms erfolgt:                                */
15 /*   pro.run Bild1 Bild2 ...                                        */
16 /* Jetzt  werden die einzelnen Bilder geladen. Tritt ein Fehler auf, */
17 /* so wird dies gemeldet. (Fehler 1: File konnte nicht richtig gela- */
18 /* den werden oder kein Speicherplatz fuer Puffer verfuegbar / Feh- */
19 /* ler 2: kein IFF-Format / Fehler 1ØØ: nicht genug Speicher) Nach- */
2Ø /* dem nun die Bilder geladen sind, wird ein Consolfenster zur      */
21 /* Steuerung und Eingabe geöffnet. Folgende Eingaben sind moeglich: */
22 /* e flagsingle       Einzelbild an (1) oder aus (Ø)                */
23 /* p flagvor,flagzu   Vorwaerts an/aus, Rueckwaerts an/aus          */
24 /* n anz              Anzahl der Durchlaeufe                        */
25 /* z zeit             Abspielrate                                   */
26 /* b bildstart,bildend Anfangs- Endbild beim Abspielen              */
27 /* a                  Ausgabe der gesetzten Parameter               */
28 /* d                  default Werte setzen                          */
29 /* s                  Film starten                                  */
3Ø /* x                  Programm verlassen                            */
31 /*************************************************************************/
32
33
34 #include <libraries/dos.h>
35 #include <exec/types.h>
36 #include <exec/exec.h>
37 #include <exec/memory.h>
38 #include <graphics/gfx.h>
39 #include <graphics/gfxmacros.h>
4Ø #include <graphics/copper.h>
```

```
#include <graphics/gels.h>
#include <graphics/regions.h>
#include <graphics/clip.h>
#include <graphics/text.h>
#include <graphics/gfxbase.h>
#include <graphics/view.h>
#include <hardware/dmabits.h>
#include <hardware/custom.h>
#include <hardware/blit.h>


#define BMHDChunk 1
#define CMAPChunk 2
#define BODYChunk 3
#define CAMGChunk 4
#define Unknown   5

#define N 100         /* moegliche Anzahl der Bilder            */
#define OUT_MEM 100

struct CMAPC          /* Definition des CMAP-Chunks             */
{
 int anz;             /* Anzahl der Farben                      */
 struct ColorRegister /* Definition einer Farbe                 */
 {
  UBYTE red;
  UBYTE green;
  UBYTE blue;
 } Color[64];         /* lieber zuviel als zuwenig              */
};

struct FORMC          /* Definition des Anfang-Chunks           */
{
 char  FORMName[4];   /* handelt es sich um ein IFF-Format?     */
 long  len;           /* Gesamte Laenge des Files               */
 char  ILBM[4];       /* handelt es sich um ILBM-Format?        */
};

struct BMHDC          /* Definition des BMHD                    */
{
 char  BMHDName[4];
 long  lenBMHD;       /* Laenge des BMHD-Chunks                 */
 short Width;         /* Breite des Bildes                      */
 short Height;        /* Hoehe des Bildes                       */
 short xpos;          /* x-Position des Bildes                  */
 short ypos;          /* y-Position des Bildes                  */
 char  Depth;         /* Anzahl der BitPlanes                   */
 char  mask;          /* Mask-Bit                               */
 char  compression;   /* Bild 'compressed'?                     */
 char  null;
 short color;
 char  xaspect;
 char  yaspect;
 short PWidth;
 short PHeight;
};

struct CHUNKID        /* hier werden die zu analysierenden Bytes */
{                     /* abgelegt                               */
 char  Name[4];       /* Name des Chunks                        */
 int   len;           /* Laenge des Chunks                      */
};

struct HEADER         /* diese Struktur enthaelt die Zeichenfolge */
{                     /* um den Chunk zu erkennen.              */
 char bmhd[4];
 char cmap[4];
 char body[4];
 char camg[4];
};

struct CAMGC          /* Definition des CAMG                    */
{
 char CAMGName[4];
 long lenCAMG;
 long ViewModes;
};

struct CMAPC CMAP1;
struct FORMC FORM1;
struct BMHDC BMHD1;
struct CHUNKID CHUNKid;
```

```
124 struct HEADER Header=
125 {
126  'B','M','H','D',    /* BMHD-Chunk */
127  'C','M','A','P',    /* CMAP-Chunk */
128  'B','O','D','Y',    /* BODY-Chunk */
129  'C','A','M','G',    /* CAMG-Chunk */
130 };
131
132 char form[]="FORM";
133 char ilbm[]="ILBM";
134
135 struct View MyView[N];
136 struct View *OldView;
137 struct ViewPort MyViewPort[N];
138 struct RasInfo MyRasInfo[N];
139 struct BitMap MyBitMap[N];
140 struct RastPort MyRastPort[N];
141 struct GfxBase *GfxBase;
142 struct DosBase *DosBase;
143
144
145 long laenge,file,end,*waste;
146 BYTE *speicher,*zeiger,*zeigerØ,err;
147 USHORT i,j,Width[N],Height[N],Depth[N],ViewModes[N],bild,
    bildanz,bildranz;
148 USHORT TopEdge[N],LeftEdge[N];
149 UBYTE *port;   /* fuer Einzelbild (s. auch frame.c) */
150 /* Anzahl der Durchlaeufe / Wartezeit / Vor-Zurueck / Einzelbild */
151 int an,anz=1,zeit=1Ø,flagvor=1,flagzu=Ø,flagsingle=Ø,bildstart,bildend;
152
153
154
155 /******************/
156 /* Haupt-Programm */
157 /******************/
158
159 main(argc,argv)
160 short argc;
161 char *argv[];
162 {
163  if(argc<2)
164  {
165     printf("%s: <Bild-File[1]> <Bild-File[2]> ...\n",argv[Ø]);
166     exit(FALSE);
167  }
168  bildanz=argc-1;
169  port=12574721;
170  open_();
171  for(bild=bildranz=Ø;bild
     <bildanz;++bild,++bildranz) /* Laden der Bilder */
172     if((err=load(argv[bild+1],bildranz))!=Ø)
173     {
174        --bildranz;
175        printf("Bild %s kann nicht geladen werden (Fehler:%d)!\n",
176               argv[bild+1],err);
177        if(err==OUT_MEM) bild=bildanz;
178     }
179  if(bildranz==Ø)
180     close_();
181  printf("\n%d Bilder geladen.\n",bildranz);
182  bildend=bildranz-1; bildstart=Ø;
183  for(;;)
184  {
185     eingabe();
186     for(an=Ø;an<anz;++an)
187     {
188        if(flagvor==1)
189        {
190           for(bild=bildstart;bild<=bildend;++bild)
191           {
192              LoadView(&MyView[bild]);
193              if(flagsingle==1)
194              {
195                 while((*port&64)==64); /* Warten bis Taste gedrueckt */
196                 Delay(1Ø);
197              }
198              else
199                Delay(zeit); /* Warten */
200           }
201        }
202        if(flagzu==1)
203        {
204           for(bild=bildend-1;bild>bildstart;--bild)
205           {
```

```
206             LoadView(&MyView[bild]);
207             if(flagsingle==1)
208             {
209                while((*port&64)==64); /* Warten bis Taste gedrueckt */
210                Delay(10);
211             }
212             else
213               Delay(zeit); /* Warten */
214          }
215       }
216    }
217    LoadView(OldView); /* alten View wieder laden */
218 }
219 close_();
220 }
221
222
223 /* Einlesen ueber Console */
224 eingabe()
225 {
226  char line[80];
227
228  if((file=Open("con:0/0/320/80/Eingabe",MODE_NEWFILE))==0)
229  {
230     printf("Console kann nicht geoeffnet werden!\n");
231     close_();
232  }
233  while(inline(line,file,80)!=0)
234  {
235     switch(line[0])
236     {
237        case 10 : /* Return-Zeile */
238                  break;
239        case 'z': /* Zeit-Definition */
240                  if(sscanf(line+2,"%d",&zeit)!=1)
241                     printf("Eingabefehler!\n");
242                  break;
243        case 'b': /* Anfangs- Endbild */
244                  if(sscanf(line+2,"%d,%d",&bildstart,&bildend)!=2)
245                     printf("Eingabefehler!\n");
246                  --bildstart,--bildend;
247                  if(bildstart<0) bildstart=0;
248                  if(bildstart>(bildranz-1)) bildstart=bildranz-1;
249                  if(bildend<0) bildend=0;
250                  if(bildend>(bildranz-1)) bildend=bildranz-1;
251                  break;
252        case 'n': /* Anzahl der Durchlaeufe */
253                  if(sscanf(line+2,"%d",&anz)!=1)
254                     printf("Eingabefehler!\n");
255                  break;
256        case 'p': /* Vor-Rueckwaerts abspielen */
257                  if(sscanf(line+2,"%d,%d",&flagvor,&flagzu)!=2)
258                     printf("Eingabefehler!\n");
259                  if(flagvor!=0) flagvor=1;
260                  if(flagzu!=0) flagzu=1;
261                  break;
262        case 'e': /* Einzelbild */
263                  if(sscanf(line+2,"%d",&flagsingle)!=1)
264                     printf("Eingabefehler!\n");
265                  if(flagsingle!=0) flagsingle=1;
266                  break;
267        case 'd': /* default Werte setzen */
268                  anz=1,zeit=10,flagvor=1,flagzu=0,flagsingle=0;
269                  bildstart=0; bildend=bildranz-1;
270                  ausgabe();
271                  break;
272        case 'a': /* Ausgabe der Parameter */
273                  ausgabe();
274                  break;
275        case 's': /* Animation starten */
276                  Close(file);
277                  return(0);
278        case 'x': /* Programm beenden */
279                  Close(file);
280                  close_();
281        default : /* Fehler aufgetreten */
282                  printf("Eingabefehler!\n");
283                  break;
284     }
285  }
286 }
287
288 inline(c,handle,len) /* eine Zeile des Files einlesen */
289 char c[];
```

```
long *handle;
short len;
{
 short i=Ø,ende=Ø;

 while((i<len)&&(c[i-1]!=1Ø)&&((ende=Read(handle,&c[i],1))!=Ø))
    ++i;
 c[i]=Ø;
 return(ende);
}

ausgabe()
{
 printf("\nParameter:\n");
 printf("  %d Bilder geladen\n",bildranz);
 printf("  Abspielen von %d nach %d\n",bildstart+1,bildend+1);
 printf("  Abspielrate:%d\n  Durchlaeufe:%d\n",zeit,anz);
 printf("  vorwaerts:");
 if(flagvor==1) printf(" an  ");
 else printf(" aus  ");
 printf("  rueckwaerts:");
 if(flagzu==1) printf(" an\n");
 else printf(" aus\n");
 printf("  Einzelbild:");
 if(flagsingle==1) printf(" an\n");
 else printf(" aus\n");
 printf("FastMem=%d ChipMem=%d\n",AvailMem(MEMF_FAST),AvailMem
 (MEMF_CHIP));
}


/*********************/
/* Initialisierungs- */
/* routinen          */
/*********************/

open_()
{
 if((GfxBase=OpenLibrary("graphics.library",Ø))==Ø)
    close_();
 if((DosBase=OpenLibrary("dos.library",Ø))==Ø)
    close_();
 OldView=GfxBase->ActiView; /* alten View retten */
}

close_()
{
 if(bildranz!=Ø)
 {
    LoadView(OldView); /* alte View-Struktur wieder initialisieren */
    for(bild=Ø;bild<bildranz;++bild)
       remove_bild(bild);
 }
 if(GfxBase!=Ø) CloseLibrary(GfxBase);
 if(DosBase!=Ø) CloseLibrary(DosBase);
 exit(Ø);
}


init_bild(n)
int n;
{
 InitView(&MyView[n]); /* View-Struktur initialisieren */
 InitVPort(&MyViewPort[n]); /* ViewPort-Struktur initialisieren */
 MyView[n].Modes=MyViewPort[n].Modes=ViewModes[n];
 MyView[n].ViewPort=&MyViewPort[n];
 MyViewPort[n].DWidth=Width[n];
 MyViewPort[n].DHeight=Height[n];
 MyViewPort[n].RasInfo=&MyRasInfo[n];
 InitBitMap(&MyBitMap[n],Depth[n],Width[n],Height[n]); /* BitMap
                                                    initialisieren */
 MyRasInfo[n].BitMap=&MyBitMap[n]; /* RasInfo-Str. initialisieren */
 MyRasInfo[n].RxOffset=LeftEdge[n];
 MyRasInfo[n].RyOffset=TopEdge[n];
 MyRasInfo[n].Next=Ø;
 /* BitMap-Speicher allocatieren */
 for(i=Ø;i<Depth[n];++i)
    if((MyBitMap[n].Planes[i]=AllocRaster(Width[n],Height[n]))==Ø)
    /* nicht genug Speicher */
    {
       printf("\n\n!!!Nicht genug Speicher!!!\n");
       for(j=Ø;j<i;++j)
          FreeRaster(MyBitMap[n].Planes[j],Width[n],Height[n]);
```

```
371         return(FALSE);
372       }
373    MakeVPort(&MyView[n],&MyViewPort[n]);
374    MrgCop(&MyView[n]);
375    for(i=Ø;i<Depth[n];++i)
376       BltClear(MyBitMap[n].Planes[i],RASSIZE(Width[n],Height[n]),Ø);
377    /* Bild geladen -> Farben setzen */
378    for(i=Ø;i<CMAPl.anz;++i)
379       SetRGB4(&MyViewPort[n],i,(short)((float)CMAPl.Color[i].red/16),
38Ø             (short)((float)CMAPl.Color[i].green/16),
381             (short)((float)CMAPl.Color[i].blue/16));
382    return(TRUE);
383 }
384
385
386 remove_bild(n)
387 int n;
388 {
389  for(i=Ø;i<Depth[n];++i)
39Ø     FreeRaster(MyBitMap[n].Planes[i],Width[n],Height[n]);
391  FreeVPortCopLists(&MyViewPort[n]);
392  FreeCprList(MyView[n].LOFCprList);
393 }
394
395
396 /*********************/
397 /* Laderoutinen      */
398 /*********************/
399
4ØØ load(fname,n)
4Ø1 char *fname;
4Ø2 int n;
4Ø3 {
4Ø4  short id;
4Ø5
4Ø6  if(init_load_file(fname)==Ø)
4Ø7  {
4Ø8     close_load_file();
4Ø9     return(1);
41Ø  }
411  end=read_file(&FORMl,sizeof(FORMl));
412  if((strncmp(&FORMl.FORMName,&form,4)!=Ø)
     (strncmp(&FORMl.ILBM,&ilbm,4)!=Ø))
413  {
414    close_load_file();
415    return(2);
416  }
417  while(end!=Ø)
418  {
419     end=read_file(&CHUNKid,sizeof(CHUNKid));
42Ø     if(end==8) id=test();
421     else {end=Ø; break;};
422     switch(id)        /* Welcher Chunk liegt vor? */
423     {
424     case BMHDChunk:
425             BMHDl.lenBMHD=CHUNKid.len;
426             end=read_file(&BMHDl.Width,CHUNKid.len);
427             if((end<CHUNKid.len)) break;
428             if(CHUNKid.len<(sizeof(BMHDl)-12))
429             {
43Ø                close_load_file();
431                return(2);
432             }
433             Width[n]=BMHDl.Width;
434             Depth[n]=BMHDl.Depth;
435             Height[n]=BMHDl.Height;
436             LeftEdge[n]=BMHDl.xpos;
437             TopEdge[n]=BMHDl.ypos;
438             if(ViewModes[n]==Ø) /* ViewMode schon gesetzt? */
439             {
44Ø                if(Width[n]>32Ø) ViewModes[n]+=HIRES;
441                if(Height[n]>25Ø) ViewModes[n]+=LACE;
442                if(Depth[n]==6) ViewModes[n]+=HAM;
443             }
444             break;
445     case CMAPChunk:
446             CMAPl.anz=CHUNKid.len/3;
447             for(i=Ø;i<CMAPl.anz;++i)
448                end=read_file(&(CMAPl.Color[i].red),3);
449             break;
45Ø     case BODYChunk:
451             if(init_bild(n)==FALSE)
452             {
```

```
453                close_load_file();
454                return(OUT_MEM);
455             }
456             get_pic(n);
457             break;
458     case CAMGChunk:
459             end=read_file(&ViewModes[n],2);
460             end=read_file(&ViewModes[n],2); /* erst hier richtig */
461             break;
462     case Unknown:
463             if((waste=AllocMem(CHUNKid.len,MEMF_CLEAR))==Ø)
464               {end=Ø; break;}
465             end=read_file(waste,CHUNKid.len);
466             FreeMem(waste,CHUNKid.len);
467             break;
468     }
469  }
470  close_load_file();
471  return(Ø);
472 }
473
474
475 /* Um welchen Chunk handelt es sich? */
476 test()
477 {
478  char *p;
479  short i;
480
481  for(i=Ø,p=&Header;
482      (i<sizeof(Header)/4)&&(strncmp(p,&CHUNKid.Name,4)!=Ø);++i,p+=4);
483  return(i+1);
484 }
485
486
487 /* Laden der BitPlanes */
488 get_pic(n)
489 int n;
490 {
491  short count;
492  char *next,*nextØ,*ne;
493
```

```
494  printf("Bild %d -> %dx%d %d Planes\n",n,Width[n],Height[n],Depth[n]);
495  printf("FastMem=%d ChipMem=%d\n",AvailMem(MEMF_FAST),
     AvailMem(MEMF_CHIP));
496  if(BMHDl.compression!=Ø)          /* File compressed */
497  {     .
498     for(i=Ø;(i<BMHDl.Height)&&(end!=Ø);++i)
499     {
500        for(j=Ø;j<BMHDl.Depth;++j)
501        {
502           nextØ=next=MyBitMap[n].Planes[j]+BMHDl.Width/8*i;
503           while((next-nextØ)<BMHDl.Width/8)
504           {
505              end=read_file(next,1);
506              if(*next==-128) continue;  /* noop */
507              if(*next>=Ø)
508              {
509                 count=1+(*next);
510                 end=read_file(next,count);
511                 next+=count;
512                 continue;
513              }
514              if((*next<Ø)&&(*next!=-128))
515              {
516                 count=1-(*next);
517                 end=read_file(next,1);
518                 for(ne=next;count>Ø;--count,*next=*ne,++next);
519                 continue;
520              }
521           }
522
523        }
524     }
525  }
526  else
527  {
528     for(i=Ø;i<BMHDl.Height;++i)
529        for(j=Ø;j<BMHDl.Depth;++j)
530           end=read_file(MyBitMap[n].Planes[j]
                        +BMHDl.Width/8*i,BMHDl.Width/8);
531  }
532 }
533
```

```
534
535 init_load_file(fname)
536 char *fname;
537 {
538  if((laenge=filelaenge_(fname))==Ø)
539     return(Ø);
54Ø  if((file=Open(fname,MODE_OLDFILE))==Ø)
541     return(Ø);
542  if((speicher=AllocMem(laenge,MEMF_CLEAR))==Ø)
543     return(Ø);
544  Read(file,speicher,laenge);
545  zeiger=zeigerØ=speicher;
546  return(1);
547 }
548
549
55Ø filelaenge_(fname)
551 char *fname;
552 {
553  struct FileInfoBlock *MyInfoBlock;
554  long *MyLock,help,l;
555
556  if((MyLock=Lock(fname,ACCESS_READ))==Ø)
557     return(Ø);
558  if((help=AllocMem(1ØØØ,MEMF_CLEAR))==Ø)
559     return(Ø);
56Ø  MyInfoBlock=help+help%4;
561  if((Examine(MyLock,MyInfoBlock))==Ø)
562  {
563     UnLock(MyLock);
564     FreeMem(help,1ØØØ);
565     return(Ø);
566  }
567  UnLock(MyLock);
568  l=MyInfoBlock->fib_Size;
569  FreeMem(help,1ØØØ);
57Ø  return(l);
571 }
572
573
574 close_load_file()
575 {
576  if(speicher!=Ø) FreeMem(speicher,laenge);
577  if(file!=Ø) Close(file);
578 }
579
58Ø
581 read_file(p,anz)
582 BYTE *p;
583 int anz;
584 {
585  while((zeiger<=speicher+laenge)&&(zeiger-zeigerØ<anz))
586  {
587     *p=*zeiger;
588     ++zeiger,++p;
589  }
59Ø  zeigerØ=zeiger;
591 }
```

Execute-File *help* zur Vereinfachung des Programmstarts:

```
pro.run picØ pic1 pic2 pic3 pic4 pic5 pic6 pic7 pic8 pic9
```

Mit der Einrichtung eines solchen Execute-Files spart man sich die ständige Angabe beim Starten des Projektors. Dieser wird dann durch `execute help` gestartet.
So, jetzt können wir Bilder abspielen. Eigentlich brauchen wir nur noch ein Programm, das eine Bildfolge generiert. Auf der beiliegenden Programmdiskette sind mehrere Programm-Versionen vorhanden, um Filmsequenzen über die Mandelbrot-Menge und die Julia-Mengen zu erstellen. An dieser Stelle möchte ich mich auf ein Programm beschränken, denn die anderen Versionen unterscheiden sich nur geringfügig.

Das folgende Programm generiert eine Bildfolge von ebenen Mandelbrot-Ausschnitten. Dabei werden beim Starten zwei Bereiche, n_{max} und die erwünschte Anzahl `anz` der Grafiken, die berechnet werden sollen, angegeben. Das Programm berechnet dann `anz`

Bilder und »zoomt« bei der Berechnung in regelmäßigen Schritten vom ersten in den zweiten Bereich. Die Bilder werden automatisch abgespeichert und gemäß ihrer Reihenfolge durchnumeriert. Das erste Bild ist `picØ`, dann folgt `pic1` usw.

Wollen wir zum Beispiel vom Bereich `[-1,2][-1.2,1.2]` in `[-.5,.5][1,1.2]` in 10 Schritten (Bildern) »zoomen«, so starten wir das Programm folgendermaßen:

```
filmPlane.run 1ØØ,1Ø,-1,2,-1.2,1.2,-.5,.5,1,1.2
```

Wir haben eine »Rechentiefe« von n_{max}=1ØØ gewählt.

Programm: *filmPlane.c*

```
/*************************************************************************/
/* filmPlane.c                                              29.12.87 */
/* Version 1.Ø                                                       */
/*                                                                   */
/*                                                                   */
/*                                                                   */
/* fuer Markt+Technik geschrieben von Heiko Knappe                   */
/*                                                                   */
/* Komplexe Dynamische Systeme                                       */
/*************************************************************************/


#include <exec/types.h>
#include <intuition/intuition.h>
#include <graphics/gfxmacros.h>

#define WIDTH   32Ø  /* Screen Modus */
#define HEIGHT  2ØØ
#define DEPTH   5
#define FLAGS   Ø
#define FWIDTH  32Ø  /* falls nur ein kleinerer Ausschnitt          */
#define FHEIGHT 2ØØ  /* berechnet werden soll.                      */
#define FXOFF   Ø
#define FYOFF   Ø

extern short save();

struct IntuitionBase *IntuitionBase;
struct GfxBase *GfxBase;
struct Screen *MyScreen;
struct RastPort *MyRastPort;
struct NewScreen MyNewScreen=
 {Ø,Ø,WIDTH,HEIGHT,DEPTH,1,Ø,FLAGS,CUSTOMSCREEN,Ø,Ø,Ø,Ø};

int   n,nmax=1ØØ;            /* Iterationsschritte                */
int   anz,bild;              /* Anzahl der Bilder                 */
float emin=.ØØ1,emax=1ØØØØØ; /* Abbruchkriterium                  */
float rmin=-1.5,rmax=2.5;    /* Grenzen des Bildes                */
float imin=-1.2,imax=1.2;
float r1min,r1max,i1min,i1max,r2min,r2max,i2min,i2max;
float rst,ist,r1st,i1st,r2st,i2st;
float re,im,re1,re2,im1,im2,cre,cim;
                             /* re,im laufender Re- und Im-Teil   */
                             /* re1,re2,im1,im2  -> Folgenglieder */
float d;                     /* Abstand zu dem Bezugspunkt        */
short x,y;                   /* Punktkoordinaten                  */
short anzcol,col,i;
char  file_name[2Ø]="pic",num[1Ø];


/***********************/
/* Hauptprogramm       */
/***********************/

main(argc,argv)
int argc;
char *argv[];
{
 if(argc!=2)
 {
    printf("Eingabe: %s <nmax,anz,r1min,r1max,i1min,i1max,",argv[Ø]);
    printf("r2min,r2max,i2min,i2max>\n");
    printf(" nmax: maximale Anzahl der Duerchlaeufe\n");
    printf(" anz:  Anzahl der Bilder\n");
    printf(" [r1min,r1max][i1min,i1max]: Wertebereich des
    1. Bildes\n");
```

```
66      printf(" [r2min,r2max][i2min,i2max]: Wertebereich
        des letzten Bildes\n");
67      close_things();
68    }
69    if(sscanf(argv[1],"%d,%d,%f,%f,%f,%f,%f,%f,%f,%f",&nmax,&anz,
70      &r1min,&r1max,&i1min,&i1max,&r2min,&r2max,&i2min,&i2max)!=10)
71    {
72      printf("falsche Eingabe\n");
73      close_things();
74    }
75    printf("\nParameter:\n emin=%f emax=%f\n",emin,emax);
76    printf(" nmax=%d\n",nmax);
77    printf(" erstes Bild:  [%f,%f] [%f,%f]\n",r1min,r1max,i1min,i1max);
78    printf(" letztes Bild: [%f,%f] [%f,%f]\n",r2min,r2max,i2min,i2max);
79    printf(" %d Bilder werden berechnet.\n",anz);
80    emin*=emin;  /* mit dem Quadrat zu vergleichen erspart Wurzeln zu */
81    emax*=emax;  /* berechnen -> Schnelligkeit                        */
82    if(anz<=1) anz=2;
83    open_things();
84    r1st=(r2max-r1max)/(anz-1),i1st=(i2max-i1max)/(anz-1);
85    r2st=(r2min-r1min)/(anz-1),i2st=(i2min-i1min)/(anz-1);
86    rmin=r1min,rmax=r1max,imin=i1min,imax=i1max;
87    for(bild=0;bild<anz;++bild,rmin+=r2st,rmax+=r1st,imin+=i2st,
      imax+=i1st)
88    {
89      itoa(bild,num);
90      strcpy(file_name+3,num);
91      printf("Berechnung Bild: %d -> %s\n      [%f,%f][%f,%f]\n",
92            bild,file_name,rmin,rmax,imin,imax);
93      mandel();
94      if(save(file_name,1,MyScreen)!=TRUE) break; /* Fehler? */
95    }
96    close_things();
97  }
98
99  /* Hilfsfunktion (int nach ASCII) */
100 itoa(wert,buff)
101 int wert;
102 char buff[];
103 {
104   int i,j,sign;
105   char c;
106
107   if((sign=wert)<0)
108     wert=-wert;
109   i=0;
110   do
111   {
112     buff[i++]=wert%10+'0';
113   }
114   while((wert/=10)>0);
115   if(sign<0)
116     buff[i++]='-';
117   buff[i]=0;
118   for(i=0,j=strlen(buff)-1;i<j;i++,j--)
119   {
120     c=buff[i];
121     buff[i]=buff[j];
122     buff[j]=c;
123   }
124 }
125
126
127 /***********************/
128 /* Initialisierung     */
129 /***********************/
130
131 open_things()
132 {
133   if((IntuitionBase=OpenLibrary("intuition.library",0))==0)
134     close_things();
135   if((GfxBase=OpenLibrary("graphics.library",0))==0)
136     close_things();
137   if((MyScreen=OpenScreen(&MyNewScreen))==0)
138     close_things();
139   MyRastPort=&(MyScreen->RastPort);
140   init_colors();
141 }
142
143 close_things()
144 {
145   if(IntuitionBase!=0) CloseLibrary(IntuitionBase);
146   if(GfxBase!=0) CloseLibrary(GfxBase);
```

```
147   if(MyScreen!=Ø) CloseScreen(MyScreen);
148   exit(Ø);
149  }
15Ø
151  init_colors()
152  {
153   for(i=Ø,anzcol=1;i<DEPTH;anzcol*=2,++i);
154   for(i=Ø;i<anzcol/2;++i)
155      SetRGB4(&(MyScreen->ViewPort),i,
156              (short)((float)i/anzcol*15*2),Ø,Ø);
157   for(i=anzcol/2;i<anzcol;++i)
158      SetRGB4(&(MyScreen->ViewPort),i,15,Ø,
159              (short)((float)(i-anzcol/2)/anzcol*15*2));
16Ø  }
161
162
163  /***********************/
164  /* Bild-Berechnung     */
165  /***********************/
166
167  mandel()
168  {
169   rst=(rmax-rmin)/FWIDTH;
17Ø   ist=(imax-imin)/FHEIGHT;
171   for(x=Ø;x<FWIDTH;++x) /* in x-Richtung */
172   {
173      re=rmin+x*rst;
174      for(y=Ø;y<FHEIGHT;++y) /* in y-Richtung */
175      {
176         /* Kontrollpunkt */
177         SetAPen(MyRastPort,anzcol-1);
178         WritePixel(MyRastPort,x+FXOFF,y+FYOFF);
179         im=imin+y*ist;
18Ø         for(n=re1=im1=Ø,cre=re,cim=im;n<nmax;++n)
181         {
182            im2=2*re1*im1-cim,re2=re1*re1-im1*im1-cre;
183            im1=im2,re1=re2;
184            d=im1*im1+re1*re1;
185            /* Ist der Attraktor erreicht? */
186            if(d<emin)  break;
187            /* Attraktor Unendlich? */
188            if(d>emax)  break;
189         }
19Ø         col=n%(anzcol-1)+1;
191         if(n==nmax) col=Ø; /* unbestimmt -> schwarz */
192         SetAPen(MyRastPort,col);
193         WritePixel(MyRastPort,x+FXOFF,y+FYOFF);
194      }
195   }
196  }
```

6 Der Fraktal-Generator – Programmbeschreibung zum Programm *fraktal.c*

Ein Programm abzudrucken hat nur Sinn, wenn es gleichzeitig gut dokumentiert wird, gerade, wenn es umfangreicher ist. Sinn und Zweck ist es ja nicht, das Programm selbst einzugeben, sondern wichtige Routinen und Programmteile in eigenen Programmen verwenden zu können. Das erfordert in erster Linie eine durchsichtige und übersichtliche Programmierweise. Ich hoffe, daß es mir gelungen ist, die in diesem Buch abgedruckten Programme derartig zu gestalten.

Wenn man anfängt, sich mit Apfelmännchen, Julia-Mengen und ihren Verwandten zu beschäftigen und versucht, sie selbst zu generieren, steht man vor dem Problem, für jede Funktion ein eigenes Programm zu schreiben. Diese Programme gleichen sich aber untereinander fast vollständig. Außerdem ist es gerade zu Beginn zuviel Arbeit, ein Programm mit komfortabler Eingabe (frame-Routine usw.) zu gestalten, so daß die Reproduktion bekannter Bilder im Vordergrund steht, da ihre Lage in der komplexen Ebene und ihr Aussehen bekannt ist.

Dieses Programm bietet die Möglichkeit komfortabel (das ist natürlich relativ!) mit elf verschiedenen Funktionen auf Entdeckungsreise zu gehen und auch eigene einzugeben. Wenn an manchen Stellen (ich denke hier an die Eingabe der Werte über ein Consol-Fenster) die Handhabung des Programms nicht optimal gelöst ist, so liegt dies an dem maximalen Umfang, der im Rahmen dieses Buches zulässig ist. Das gesamte Programm ist selbst jetzt bereits über 30 Kbyte lang. Sicher ist jedenfalls, daß ohne große Klimmzüge das Verhalten komplexer Rückkopplungen untersucht werden kann. Natürlich kann das Programm nicht alles! Für spezielle Untersuchungen wird eine eigene Programmierung sicher nicht zu umgehen sein, wobei die Verwendung einiger Programmteile sicher auch hilfreich ist.

Der Source besteht aus zwei Teilen. Der erste *(fraktal.menu)* wird in das eigentliche Programm als Include-File eingebunden. Er enthält alle Definitionen für die Menüs, das Gadget und alle Variablen. Die Abtrennung der Vordefinitionen vom Hauptprogramm fördert die Übersicht enorm und sollte deshalb bei längeren Programmen immer vorgenommen werden. Die Einbindung ins Hauptprogramm geschieht mit `#include "Fraktal.menu"`. Damit der Compiler keinen Fehler meldet, muß in diesem Fall das File in dem Directory stehen, in dem auch die anderen C-Include-Files vorhanden sind. Soll der Compiler das File zum Beispiel aus dem RAM verwenden, so muß beim Aztec C `#include<ram:Fraktal.menu>` im Hauptprogramm stehen. Beim Lattice-Compiler können Directories angegeben werden (Option `-i`), in denen nach Include-Files gesucht werden soll. (Damit der Compiler nach dem einzubindenden File auch in den angegeben Directories sucht, muß der Include-Befehl

`#include"..."` und nicht `#include<..>` lauten!)

Noch eine Bemerkung zum Kompilieren mit dem Aztec C: Normalerweise rechnet der Aztec C nur mit *float* und nicht mit *double* Variablen (auch wenn diese so definiert werden), was zwar Schnelligkeit bringt, für uns aber den Nachteil hat, daß wir nicht sehr tief in die Ebene eindringen können. Diesen Nachteil der geringeren Rechengenauigkeit können wir durch eine Kompiler-Option und die Verwendung eines anderen Mathematik-Libraries wettmachen.

Um den Kompiler zur Verwendung der genaueren Mathematik-Routinen zu veranlassen, muß mit der Option »`+fi`« kompiliert werden (`+ff` default). Vorher sind alle *float*-Variablen als *double* zu definieren und, um ein richtiges Arbeiten von `printf` und `sscanf` zu erreichen, `%f` durch `%lf` zu ersetzen. Damit der Linker nun richtig arbeiten kann, muß außerdem noch ein anderes Mathe-Library verwendet werden: Statt bisher mit *m32.lib* zu linken, ist nun *mx32.lib* zu verwenden.

Startet man jetzt das Programm, so bemerkt man sofort die wesentlich geringere Rechengeschwindigkeit (vergleichbar mit der Geschwindigkeit eines mit Lattice C kompilierten Programms). Deshalb ist es sicher erst sinnvoll, auf die höhere Rechengenauigkeit überzugehen, wenn es wirklich unumgänglich ist.

Starten wir das Programm, so erscheint im oberen Teil des Workbench-Screens ein Fenster mit Menüs und einem Text-Gadget. In dem Text-Gadget wird der Name des Bildes für Diskoperationen, also zum Laden und Speichern, eingegeben. Mit den Menüs wird das Programm gesteuert.

Erklärung der Haupt-Menüs

Projekt-Menü

Laden: Es wird ein Bild mit dem im Text-Gadget angegebenen Namen geladen. Handelt es sich um ein mit diesem Programm gespeichertes Bild (es existiert ein »*.info*«-File mit den Bild-Parametern), so kann es weiter bearbeitet werden. Ansonsten wird es geladen und bleibt solange sichtbar, bis der Screen heruntergezogen wird (siehe Programm *loaddemo.c)*. Liegt kein ILBM-Format vor oder hat das »*.info*«-File eine falsche Struktur, so wird der Screen kurz hell, um den Fehler mitzuteilen.

Start: Startet die Berechnung eines Bildes in dem gewählten Bild-Format der gewählten Funktion und mit den vorgewählten Parametern.

Input: Öffnet ein Consol-Fenster über das die Parameter des Bildes eingegeben werden. Die Eingabe geschieht folgendermaßen:

`a`	Ausgabe der Parameter im CLI-Fenster
`x`	Beenden der Eingabe
`r rmin,rmax`	Werte der linken und rechten Bildbegrenzung (Wertebereich des Realteils von `z` bzw. `c`)
`i imin,imax`	Werte der oberen und unteren Bildbegrenzung (Wertebereich des Imaginärteils von `z` bzw. `c`)
`c cre,cim`	Real- und Imaginärteil von `c` (Julia-Mengen)
`e emin,emax`	Abbruchkriterien
`n nmax,noff`	maximale Anzahl der Durchläufe, unterdrückte Durchläufe
`v vrmin,vrmax,vimin,vimax`	bestimmt einen Bildausschnitt

(View), der markiert werden kann.

`g gridstep` Gitterunterteilungen

`d` Rücksetzen der Parameter auf Default-Werte

Ende: Beendet das Programm (analog Anklicken des Close-Gadgets).

Format-Menü

(Programm verwendet die höhere PAL-Auflösung.)

`320x250`: Format 320x250 mit 5 Planes (32 Farben) bzw. 6 Planes (64 Farben) bei Wahl von Brite

`640x250`: Format 640x250 mit 4 Planes (16 Farben)

`320x500`: Format 320x500 (interlace) mit 5 Planes (32 Farben) bzw.6 Planes (64 Farben) bei Wahl von Brite

`640x500`: Format 640x500 (interlace) mit 4 Planes (16 Farben)

Funktion-Menü

Folgende Funktionen sind bereits möglich:

1. Auswertung der komplexen Nullstellensuche mit dem Newton-Verfahren an den Funktionen:

   $z^3-1=0$,
   $z^4-1=0$
   und $z^5-1=0$.

2. Untersuchung der Funktion $z_{(n)}{}^2-c=z_{(n+1)}$, sowohl mit $z_{(0)}=0$, c=Bildpunkt (Apfelmännchen), als auch mit $z_{(0)}$=Bildpunkt, c fest gewählt (Julia-Menge).

3. Analog zum Punkt 2 die Untersuchung der Funktionen

   $z_{(n)}{}^3-c=z_{(n+1)}$
   $z_{(n)}{}^4-c=z_{(n+1)}$
   und $z_{(n)}{}^5-c=z_{(n+1)}$.

Außerdem ist ein Item Erw. vorgesehen, um ohne großen Aufwand eine eigene Funktion zu realisieren.

Extra-Menü

Brite: Schaltet die Verwendung des Half-Brite-Modus bei der Auflösung 320x250 und 320x500 ein bzw. aus. Bei den anderen beiden Auflösungen wird dies nicht berücksichtigt, da hier der Modus nicht möglich ist. Daß der Modus eingeschaltet ist, ist an dem Haken zu erkennen.

3-D: Wählt die 3-D-Interpretation des Bildes. Die Anzahl der Durchläufe (n) bestimmt die Höhe des Gebirges. Da die Anzahl der Farben begrenzt ist, wiederholen sich bei der ebenen Interpretation die Farben periodisch. Die 3-D-Interpretation ermöglicht dagegen eine wesentlich höhere Auflösung.

normal: Anzahl der Durchläufe werden nach oben aufgetragen. (n_{max} entspricht »Bergplateau«)

invertiert: Anzahl der Durchläufe werden nach unten aufgetragen. (n_{max} entspricht »Talboden«)

Wird die neue Berechnung eines Bildes gestartet, so öffnet das Programm einen neuen Screen mit einem Fenster, in dem gezeichnet wird. Der Screen erscheint unter dem Hauptfenster, damit weiterhin eine Eingabe eines Bildnamens möglich bleibt. Das Menü des Hauptfensters ist nun abgeschaltet und das neue Arbeitsfenster und dessen Menüs sind aktiv.

Erklärung des Arbeits-Menüs

Projekt-Menü

Speichern: Speichert ein Bild mit dem angegebenen Namen. Das Bildfile wird mit einem *.info*-File abgespeichert, das alle Bildparameter enthält. Damit ist es möglich, ein Bild nachträglich nach neuem Laden weiterzubearbeiten, bzw. ein noch nicht fertiges Bild abzuspeichern, das nach erneutem Laden automatisch weiterberechnet wird.

Cancel: Bricht die Bearbeitung des Bildes ab. Der Screen wird geschlossen und die Kontrolle wieder dem Hauptfenster übergeben.

Extra-Menü

Seite?: Um den verdeckten Teil des Bildes auch zu sehen, wird dieses Item angewählt. Nun wird der Screen automatisch nach oben verschoben, so daß das ganze Bild sichtbar wird. Um wieder eine Eingabe in das Text-Gadget vornehmen zu können, wird das Item erneut angewählt und der Screen verschiebt sich wieder unter das Hauptfenster. (Um das Gadget zu erreichen, kann natürlich auch durch Drücken der linken *Amiga*-Taste und »M« bzw. »N« der Workbench-Screen nach vorn bzw. nach hinten gelegt werden.)

B&W: Setzt die Farbregister abwechselnd auf schwarz und weiß. Diese Farbinterpretation ermöglicht manchmal eine bessere Interpretation der Strukturen. Um wieder ein farbiges Bild zu erhalten, muß wiederum dieses Item angewählt werden.

Grid: Legt ein Gitter im *Complement*-Zeichenmodus über das Bild. Wie viele Unterteilungen das Gitter hat, kann mit »g« bei der Eingabe über das Consol-Fenster bestimmt werden. Die Berechnung wird solange angehalten, bis die linke Maustaste gedrückt wird. Dann verschwindet das Gitter und das Bild wird weiter berechnet. Das Gitter hat den Sinn, besser abschätzen zu können, wie weit die Berechnung fortgeschritten ist, bzw. hilft es bei der ungefähren Festellung von Punktkoordinaten.

View: Zeichnet (soweit das bestimmte Gebiet vollständig im Bild liegt) eine Fläche im *Complement*-Zeichenmodus über das entsprechende Gebiet. Auch hier wird die Bearbeitung solange eingestellt, bis die linke Maustaste gedrückt wird. Die Eckkoordinaten des Gebiets können über das Consol-Fenster eingegeben werden. Sinn dieser Funktion ist es, einen Auschnitt hervorzuheben, um zum Beispiel erkennbar zu machen, welcher Ausschnitt des Bildes ein anderes darstellt.

Außer der Eingabe der Koordinaten des Bildbereichs über das Consol-Fenster kann ein interessierender Bereich direkt gewählt werden. Dazu drückt man die linke Maustaste nachdem die Maus über die linke obere Ecke des Bereichs positioniert wurde. Solange die Maustaste gedrückt wird, kann durch Verschieben der Maus ein Gebiet eingegrenzt werden. Sobald die Maustaste losgelassen wird, werden die Werte der Gebietsgrenzen berechnet, das Bild gelöscht und die Berechnung neu gestartet. Wurde versehentlich die Maustaste gedrückt, so verhindert man die Markierung des Bereichs, indem man die Maus links bzw. oberhalb des Anfangspunktes wieder losläßt. Wird ein zu kleiner Bereich angewählt (siehe Programm *frame.c*), so wird er auch nicht übernommen.

Noch eine Bemerkung zum Menü des Fensters. Um beim Speichern sicherzustellen, daß das Bild richtig abgelegt wird, wurde in der NewWindow-Struktur des Arbeitsfen-

sters das *Menuverify*-Flag gesetzt. Dies bedeutet, daß nach dem Drücken der rechten Maustaste solange sämtliche Operationen (also zum Beispiel Mausbewegungen etc.) angehalten werden, bis die neue Meldung vom Programm übernommen wurde.

Dies hat zur Folge, daß zum Beispiel beim Drücken der rechten Maustaste während dem Speichern die Maus solange hängenbleibt, bis fertig gespeichert wurde.

Also keine Angst, der Amiga ist dann nicht abgestürzt!

Das Einbinden einer eigenen Funktion

Um jedem die Möglichkeit zu geben, ohne großen Aufwand das Programm zur Untersuchung eigener Problemstellungen zu nutzen, wurde, wie bereits angesprochen, eine Erweiterungsfunktion vorgesehen.

Diese Funktion `erweiterung()` steht im Programm ab Zeile 1166. Wird die Funktion *Erw.* im Menü angewählt, so wird nach dem Starten diese Routine zur Berechnung aufgerufen.

Das Programm erwartet, daß in den Funktionsroutinen die Anzahl der benötigten Durchläufe `n` bis zu einem Abbruchkriterium berechnet wird.

Als Beispiel für eine solche Erweiterungsfunktion einen »verzerrten« Schnitt durch die Folge z^2-c. Dazu setzen wir zu Beginn jeder Berechnung die Imaginärteile von `z` und `c` zum Beispiel auf `0.5`. Die Realteile werden durch die augenblicklichen Bildpunktkoordinaten (Variablen `re` und `im`) bestimmt.

```
erweiterung()
{
 for(n=Ø,im1=cim=Ø.5,cre=im,re1=re,n<nmax;++n)
 {
    im2=2*re1*im1-cim,re2=re1*re1-im1*im1-cre;
    im1=im2,re1=re2;
    d[Ø]=im1*im1+re1*re1;
    /* Ist der Attraktor erreicht? */
    if(d[Ø]<emin)  break;
    /* Attraktor Unendlich?       */
    if(d[Ø]>emax)  break;
 }
}
```

Haben wir diese Zeilen eingefügt und das Programm kompiliert (nicht vergessen die Module *iff.o* und *frame.o* an das Programm zu linken), kann's schon losgehen. Wir wählen den Bildbereich, die Auflösung, den Anzeigemodus und die Funktion *Erw.* an. Starten wir jetzt das Programm, wird das Bild unter Verwendung unserer eigenen Funktion gezeichnet.

Noch eine Anmerkung zu den Farben beim Extra-Half-Brite-Modus: Sicher werden sich einige fragen, weshalb bei diesem Modus die Farbe so »poppig« gewählt wurden. Dies ist darauf zurückzuführen, daß es nur bei geschickter Wahl der Farben möglich ist, wirklich 64 verschiedene zu erhalten. Es ist übrigens sinnvoll, die Bilder, wenn möglich, also in den beiden LoRes-Auflösungen, in diesem Modus zeichnen zu lassen. Die Rechenzeit steigt natürlich nicht an und die guten Grafikprogramme wie Deluxe Paint laden auch Bilder mit mehr als fünf Planes. Dabei wird zwar die sechste Plane weggelassen, aber die Farben dieser mit eingerechnet. Wir erhalten dann das gleiche Bild, als wenn wir ohne Half-Brite-Modus gearbeitet hätten.

Programm: *fraktal.c*

```
/*********************************************************************/
/* fraktal.menu                                          Ø3.Ø1.88 */
/* Version 2.Ø                                                    */
/*                                                                */
/* Include-File fuer das Programm Fraktal.c                       */
/*                                                                */
/* Dieses File enthaelt alle notwendigen Menue-, Gadget- und Window- */
/* Strukturen. Im Programm Fractal.c wird es durch '#include< .inc>'*/
/* eingebunden. Es ist natuerlich auch moeglich diese Definitionen  */
/* im Hauptprogramm  vorzunehmen. Die Trennung in zwei Files ist    */
/* aber in Hinblick auf Modularitaet und Uebersicht sicher sinnvoll. */
/*********************************************************************/


/* Definitionen zum Ein- und Ausschalten der Menues */
#define ON ITEMTEXTITEMENABLEDHIGHCOMPCHECKITCHECKED
#define OFF ITEMTEXTITEMENABLEDHIGHCOMP
#define LORES Res1.Flags
#define MEDRES Res2.Flags
#define HILORES Res3.Flags
#define HIMEDRES Res4.Flags
#define F1 Funk1.Flags
#define F2 Funk2.Flags
#define F3 Funk3.Flags
#define F4 Funk4.Flags
#define F5 Funk5.Flags
#define F6 Funk6.Flags
#define F7 Funk7.Flags
#define F8 Funk8.Flags
#define F9 Funk9.Flags
#define F1Ø Funk1Ø.Flags
#define F11 Funk11.Flags
#define F12 Funk12.Flags
#define DREIDN DDN.Flags
#define DREIDI DDI.Flags
#define BRITE Brite.Flags
#define BW Bw.Flags
#define PAGE Page.Flags
#define ANZFUNK 12      /* Wieviele Funktionen sind moeglich */
#define WIDTH MyNewScreen.Width=NewWorkWindow.Width
#define HEIGHT MyNewScreen.Height=NewWorkWindow.Height
#define DEPTH MyNewScreen.Depth
#define TOPEDGE MyNewScreen.TopEdge
#define FLAGS MyNewScreen.ViewModes
#define MYFIRSTMENU ProjMenu
#define WORKMENU WorkMenu


/* Text-Gadget */
UBYTE F[8Ø]="Newton-pic";
UBYTE File[8Ø]="Newton-pic";
UBYTE Fileh[8Ø]="Newton-pic.info";
struct StringInfo MyGInfo={&F,Ø,Ø,8Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø};
USHORT MyVec[]={Ø,Ø,579,Ø,579,9,Ø,9,Ø,Ø};
struct Border MyBorder={-2,-1,3,Ø,JAM1,5,MyVec,Ø};
struct Gadget Text_Gad=
 {Ø,31,12,576,8,GADGHCOMP,RELVERIFY,STRGADGET,&MyBorder,Ø,Ø,Ø,&MyGInfo,4,Ø};


/* Extra-Menu */
struct IntuiText DDIText={3,1,COMPLEMENT,2Ø,1,Ø," invertiert",Ø};
struct MenuItem DDI=
 {Ø,72,9,12Ø,1Ø,ITEMTEXT+ITEMENABLED+HIGHCOMP,Ø,&DDIText,Ø,Ø,Ø,ØxFFFF};
struct IntuiText DDNText={3,1,COMPLEMENT,2Ø,1,Ø," normal",Ø};
struct MenuItem DDN=
 {&DDI,72,-2,12Ø,1Ø,ITEMTEXT+ITEMENABLED+HIGHCOMP,Ø,&DDNText,Ø,Ø,Ø,ØxFFFF};
struct IntuiText DDText={3,1,COMPLEMENT,2Ø,1,Ø,' 3-D",Ø};
struct MenuItem DD=
 {Ø,Ø,11,76,1Ø,ITEMTEXT+ITEMENABLED+HIGHCOMP,Ø,&DDText,Ø,Ø,&DDN,ØxFFFF}
struct IntuiText BriteText={3,1,COMPLEMENT,2Ø,1,Ø,"Brite",Ø};
struct MenuItem Brite=
 {&DD,Ø,Ø,76,1Ø,ITEMTEXT+ITEMENABLED+HIGHCOMP,Ø,&BriteText,Ø,Ø,Ø,ØxFFFF};
struct Menu ExMenu={Ø,24Ø,Ø,75,Ø,MENUENABLED,"Extra",&Brite};


/* Funktions-Menu */
struct IntuiText FunkText12={3,1,COMPLEMENT,2Ø,1,Ø,"  Erw.",Ø};
struct MenuItem Funk12=
 {Ø,Ø,121,76,1Ø,ITEMTEXT+ITEMENABLED+HIGHCOMP,Ø,&FunkText12,Ø,Ø,Ø,ØxFFFF};
struct IntuiText FunkText11={3,1,COMPLEMENT,2Ø,1,Ø,"J z5-c",Ø};
struct MenuItem Funk11=
 {&Funk12,Ø,11Ø,76,1Ø,ITEMTEXT+ITEMENABLED+HIGHCOMP,Ø,&FunkText11,Ø,Ø,Ø,ØxFFFF};
```

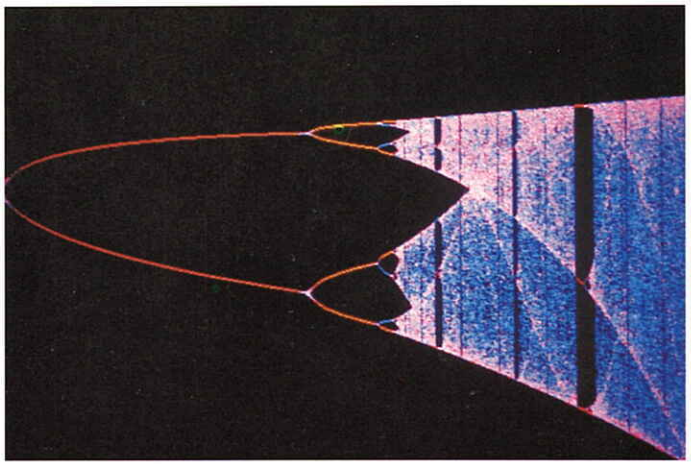
```
struct IntuiText FunkText1Ø={3,1,COMPLEMENT,2Ø,1,Ø,"J z4-c",Ø};
struct MenuItem Funk1Ø=
 {&Funk11,Ø,99,76,1Ø,ITEMTEXT+ITEMENABLED+HIGHCOMP,Ø,&FunkText1Ø,Ø,Ø,Ø,ØxFFFF};
struct IntuiText FunkText9={3,1,COMPLEMENT,2Ø,1,Ø,"J z3-c",Ø};
struct MenuItem Funk9=
 {&Funk1Ø,Ø,88,76,1Ø,ITEMTEXT+ITEMENABLED+HIGHCOMP,Ø,&FunkText9,Ø,Ø,Ø,ØxFFFF};
struct IntuiText FunkText8={3,1,COMPLEMENT,2Ø,1,Ø,"Julia",Ø};
struct MenuItem Funk8=
 {&Funk9,Ø,77,76,1Ø,ITEMTEXT+ITEMENABLED+HIGHCOMP,Ø,&FunkText8,Ø,Ø,Ø,ØxFFFF};
struct IntuiText FunkText7={3,1,COMPLEMENT,2Ø,1,Ø,"z5-c=z",Ø};
struct MenuItem Funk7=
 {&Funk8,Ø,66,76,1Ø,ITEMTEXT+ITEMENABLED+HIGHCOMP,Ø,&FunkText7,Ø,Ø,Ø,ØxFFFF};
struct IntuiText FunkText6={3,1,COMPLEMENT,2Ø,1,Ø,"z4-c=z",Ø};
struct MenuItem Funk6=
 {&Funk7,Ø,55,76,1Ø,ITEMTEXT+ITEMENABLED+HIGHCOMP,Ø,&FunkText6,Ø,Ø,Ø,ØxFFFF};
struct IntuiText FunkText5={3,1,COMPLEMENT,2Ø,1,Ø,"z3-c=z",Ø};
struct MenuItem Funk5=
 {&Funk6,Ø,44,76,1Ø,ITEMTEXT+ITEMENABLED+HIGHCOMP,Ø,&FunkText5,Ø,Ø,Ø,ØxFFFF};
struct IntuiText FunkText4={3,1,COMPLEMENT,2Ø,1,Ø,"z2-c=z",Ø};
struct MenuItem Funk4=
 {&Funk5,Ø,33,76,1Ø,ITEMTEXT+ITEMENABLED+HIGHCOMP,Ø,&FunkText4,Ø,Ø,Ø,ØxFFFF};
struct IntuiText FunkText3={3,1,COMPLEMENT,2Ø,1,Ø,"z5-1=Ø",Ø};
struct MenuItem Funk3=
 {&Funk4,Ø,22,76,1Ø,ITEMTEXT+ITEMENABLED+HIGHCOMP,Ø,&FunkText3,Ø,Ø,Ø,ØxFFFF};
struct IntuiText FunkText2={3,1,COMPLEMENT,2Ø,1,Ø,"z4-1=Ø",Ø};
struct MenuItem Funk2=
 {&Funk3,Ø,11,76,1Ø,ITEMTEXT+ITEMENABLED+HIGHCOMP,Ø,&FunkText2,Ø,Ø,Ø,ØxFFFF};
struct IntuiText FunkText1={3,1,COMPLEMENT,2Ø,1,Ø,"z3-1=Ø",Ø};
struct MenuItem Funk1=
 {&Funk2,Ø,Ø,76,1Ø,CHECKIT+ITEMTEXT+ITEMENABLED+HIGHCOMP+CHECKED,
  Ø,&FunkText1,Ø,Ø,Ø,Ø,ØxFFFF};
struct Menu FunkMenu={&ExMenu,153,Ø,84,Ø,MENUENABLED,"Funktion",&Funk1};

/* Res.-Menu */
struct IntuiText ResText4={3,1,COMPLEMENT,2Ø,1,Ø,"64Øx5ØØ",Ø};
struct MenuItem Res4=
 {Ø,Ø,33,76,1Ø,ITEMTEXT+ITEMENABLED+HIGHCOMP,Ø,&ResText4,Ø,Ø,Ø,ØxFFFF};
struct IntuiText ResText3={3,1,COMPLEMENT,2Ø,1,Ø,"32Øx5ØØ",Ø};
struct MenuItem Res3=
 {&Res4,Ø,22,76,1Ø,ITEMTEXT+ITEMENABLED+HIGHCOMP,Ø,&ResText3,Ø,Ø,Ø,ØxFFFF};
struct IntuiText ResText2={3,1,COMPLEMENT,2Ø,1,Ø,"64Øx25Ø",Ø};
struct MenuItem Res2=
 {&Res3,Ø,11,76,1Ø,ITEMTEXT+ITEMENABLED+HIGHCOMP,Ø,&ResText2,Ø,Ø,Ø,ØxFFFF};
struct IntuiText ResText1={3,1,COMPLEMENT,2Ø,1,Ø,"32Øx25Ø",Ø};
struct MenuItem Res1=
 {&Res2,Ø,Ø,76,1Ø,CHECKIT+ITEMTEXT+ITEMENABLED+HIGHCOMP+CHECKED,
  Ø,&ResText1,Ø,Ø,Ø,Ø,ØxFFFF};
struct Menu ResMenu={&FunkMenu,81,Ø,66,Ø,MENUENABLED,"Format",&Res1};

/* Projekt-Menu */
struct IntuiText EndeText={3,1,COMPLEMENT,1,1,Ø,"Ende",Ø};
struct MenuItem Ende=
 {Ø,Ø,33,113,1Ø,ITEMTEXT+ITEMENABLED+HIGHCOMP,Ø,&EndeText,Ø,Ø,Ø,ØxFFFF};
struct IntuiText InputText={3,1,COMPLEMENT,1,1,Ø,"Input",Ø};
struct MenuItem Inp=
 {&Ende,Ø,22,113,1Ø,ITEMTEXT+ITEMENABLED+HIGHCOMP,Ø,&InputText,Ø,Ø,Ø,ØxFFFF};
struct IntuiText StartText={3,1,COMPLEMENT,1,1,Ø,"Start",Ø};
struct MenuItem Start=
 {&Inp,Ø,11,113,1Ø,ITEMTEXT+COMMSEQ+ITEMENABLED+HIGHCOMP,
  Ø,&StartText,Ø,'R',Ø,ØxFFFF};
struct IntuiText LadenText={3,1,COMPLEMENT,1,1,Ø,"Laden",Ø};
struct MenuItem Laden=
 {&Start,Ø,Ø,113,1Ø,ITEMTEXT+COMMSEQ+ITEMENABLED+HIGHCOMP,
  Ø,&LadenText,Ø,'L',Ø,ØxFFFF};
struct Menu ProjMenu={&ResMenu,Ø,Ø,75,Ø,MENUENABLED,"Projekt",&Laden};

/* Definition der Menues fuer das WorkWindow */
struct IntuiText ViewText={3,1,COMPLEMENT,1,1,Ø,"  View",Ø};
struct MenuItem View=
 {Ø,Ø,33,9Ø,1Ø,ITEMTEXT+ITEMENABLED+HIGHCOMP,Ø,&ViewText,Ø,Ø,Ø,ØxFFFF};
struct IntuiText GridText={3,1,COMPLEMENT,1,1,Ø,"  Grid",Ø};
struct MenuItem Grid=
 {&View,Ø,22,9Ø,1Ø,ITEMTEXT+ITEMENABLED+HIGHCOMP,Ø,&GridText,Ø,Ø,Ø,ØxFFFF};
struct IntuiText BwText={3,1,COMPLEMENT,1,1,Ø,"  B&W",Ø};
struct MenuItem Bw=
 {&Grid,Ø,11,9Ø,1Ø,ITEMTEXT+ITEMENABLED+HIGHCOMP,Ø,&BwText,Ø,Ø,Ø,ØxFFFF};
struct IntuiText PageText={3,1,COMPLEMENT,1,1,Ø,"  Seite?",Ø};
struct MenuItem Page=
 {&Bw,Ø,Ø,9Ø,1Ø,ITEMTEXT+ITEMENABLED+HIGHCOMP,Ø,&PageText,Ø,Ø,Ø,ØxFFFF};
struct Menu ExtraMenu={Ø,81,Ø,75,Ø,MENUENABLED,"Extra",&Page};
struct IntuiText CancelText={3,1,COMPLEMENT,1,1,Ø,"Cancel",Ø};
```

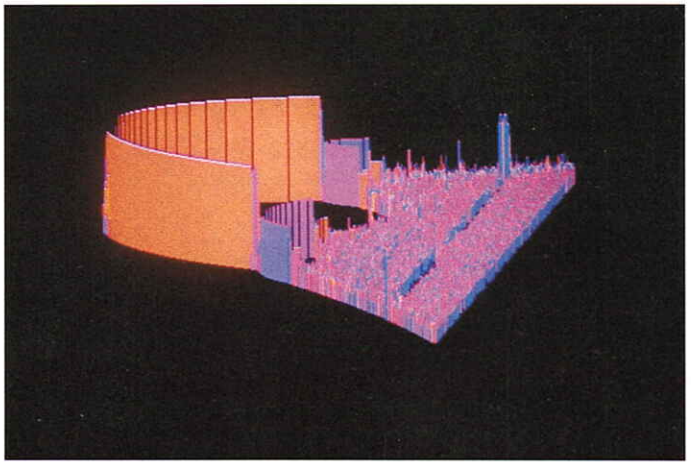
```
struct MenuItem Cancel=
 {Ø,Ø,11,113,1Ø,ITEMTEXT+COMMSEQ+ITEMENABLED+HIGHCOMP,
  Ø,&CancelText,Ø,'C',Ø,ØxFFFF};
struct IntuiText SpeichernText={3,1,COMPLEMENT,1,1,Ø,"Speichern",Ø};
struct MenuItem Speichern=
 {&Cancel,Ø,Ø,113,1Ø,ITEMTEXT+COMMSEQ+ITEMENABLED+HIGHCOMP,
  Ø,&SpeichernText,Ø,'S',Ø,ØxFFFF};
struct Menu WorkMenu={&ExtraMenu,Ø,Ø,75,Ø,MENUENABLED,"Projekt",&Speichern};


/* Window-Definition */
/* WORKBENCH Window */
struct NewWindow MyNewWindow=
 {Ø,Ø,64Ø,23,Ø,1,CLOSEWINDOW+MENUPICK+GADGETUP+GADGETDOWN,
  WINDOWDEPTH+WINDOWCLOSE+ACTIVATE+NOCAREREFRESH,&Text_Gad,Ø,
  "Komplexe Dynamische Systeme",Ø,Ø,5,5,64Ø,2ØØ,WBENCHSCREEN};
/* Arbeits-Window */
struct NewWindow NewWorkWindow=
 {Ø,Ø,Ø,Ø,15,31,MOUSEBUTTONS+MENUPICK+MENUVERIFY,GIMMEZEROZERO+BORDERLESS+
  ACTIVATE+NOCAREREFRESH,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,CUSTOMSCREEN};

/* weitere wichtige Struktur-Definitionen */
struct IntuitionBase *IntuitionBase;
struct GfxBase *GfxBase;
struct Window *MyWindow;
struct Window *WorkWindow;
struct IntuiMessage *MyMessage;
struct IntuiMessage *WorkMessage;
struct Screen *MyScreen;
struct NewScreen MyNewScreen={Ø,25,Ø,Ø,Ø,Ø,1,Ø,CUSTOMSCREEN,Ø,Ø,Ø,Ø};
struct RastPort *rp;

/* Definition des Info-Chunks */
struct INFOC          /* Info-Datei                    */
{
 float rmin;          /* xmin,xmax,ymin,ymax           */
 float rmax;          /* rel. Koordinaten des Bildes   */
 float imin;
 float imax;
 float cre;           /* cre, cim fuer Julia-Mengen    */
 float cim;
 short mx;            /* mx,my augenblickliche Koordinaten */
 short my;
 int   nmax;          /* Bild-Parameter        */
 int   noff;
 float emin,emax;
 char  mode;          /* Berechnungsmodus?     */
 char  res;           /* Aufloesung des Bildes */
 char  dim;           /* 3D Bild?              */
 char  half_brite;    /* HALF_BRITE Modus?     */
 char  text[256];
} MyInfo;

```

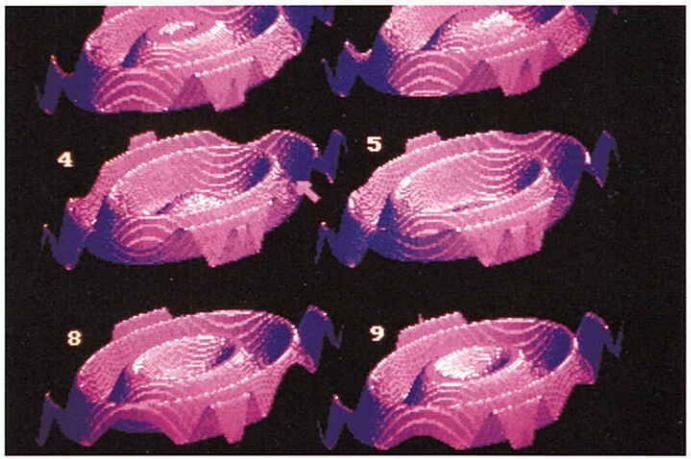
```
/*************************************************************************/
/* Fraktal.c                                                  27.Ø1.88   */
/* Version 4.Ø                                                           */
/*                                                                       */
/*                                                                       */
/* 1. Auswertung einer komplexen Nullstellen-Suche an Hand der Funktionen */
/*    (3)  z^5-1=Ø                                                       */
/*    (2)  z^4-1=Ø                                                       */
/*    (1)  z^3-1=Ø                                                       */
/* Untersuchung eines Attraktors  mit  entsprechender Farbinterpretation. */
/* (Farbe wird nach Anzahl der notwendigen Durchlaeufen bis zum Erreichen */
/* des Attraktors errechnet.)                                            */
/*    (3)   Attraktor z=.3Ø9Ø+i*.951Ø                                    */
/*          Attraktor z=-.8Ø9Ø+i*.5877                                   */
/*          Attraktor z=-.8Ø9Ø-i*.5877                                   */
/*          Attraktor z=.3Ø9Ø-i*.951Ø                                    */
/*          Attraktor z=1                                                */
/*    (2)   Attraktor z=1                                                */
/*          Attraktor z=i                                                */
/*          Attraktor z=-1                                               */
/*          Attraktor z=-i                                               */
/*    (1)   Attraktor z=-.5+i*.866                                       */
/*          Attraktor z=-.5-i*.866                                       */
/*          Attraktor z=1                                                */
/*                                                                       */
/* 2. Auswertung der komplexen Folge z^2-c=z                             */
/*    z(Ø)=Ø,c Bildpunkt -> Mandelbrodt-Ebene (Apfelmaennchen)           */
/*                                                                       */
```

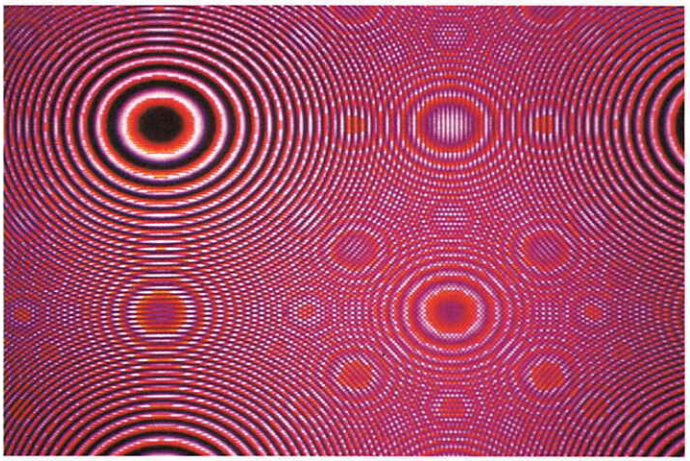
```
/* 3. Auswertung der komplexen Folge z^3-c=z                              */
/*    z(0)=0,c Bildpunkt                                                  */
/*                                                                        */
/* 4. Auswertung der komplexen Folge z^4-c=z                              */
/*    z(0)=0,c Bildpunkt                                                  */
/*                                                                        */
/* 5. Auswertung der komplexen Folge z^5-c=z                              */
/*    z(0)=0,c Bildpunkt                                                  */
/*                                                                        */
/* 6. Auswertung der komplexen Folge z^2-c=z                              */
/*    z(0) Bildpunkt, c fest -> Julia-Menge                               */
/*                                                                        */
/* 7. Auswertung der komplexen Folge z^3-c=z                              */
/*    z(0) Bildpunkt, c fest                                              */
/*                                                                        */
/* 8. Auswertung der komplexen Folge z^4-c=z                              */
/*    z(0) Bildpunkt,c fest                                               */
/*                                                                        */
/* 9. Auswertung der komplexen Folge z^5-c=z                              */
/*    z(0) Bildpunkt, c fest                                              */
/*                                                                        */
/*                                                                        */
/* fuer Markt+Technik geschrieben von Heiko Knappe                        */
/*                                                                        */
/* Komplexe Dynamische Systeme                                            */
/**************************************************************************/


#include <exec/types.h>
#include <intuition/intuition.h>
/* Einbinden des Definitionsfiles */
#include "Fraktal.menu"

#define MIN 1e-10          /* kritischer Punkt? */

/* Variablen fuer die Uebergabe der frame-Werte  */
extern float rxmin,rxmax,rymin,rymax;

/* Einbinden der externen Funktionen */
extern short save(),load(),frame();

char  funktion=0,resolution=0;/* welche Funktion? welche Aufloesung? */
char  anzcol;      /* Anzahl der Farben in der jeweiligen Aufloesung */
int   i;
float r1,r2,i1,i2;            /* Real- und Imaginaerteil der Folge   */
int   n,nmax=1000,noff=0;     /* Iterationsschritte                  */
float emin=.001,emax=1000;    /* Abbruchkriterium                    */
float rmin=-1.5,rmax=1.5;     /* Grenzen des Bildes                  */
float imin=-1.5,imax=1.5;
float vrmin,vrmax,vimin,vimax;/* ViewPort-Definition                 */
float nsre[5],nsim[5];        /* Nullstellen                         */
float rst,ist;                /* Schrittweite                        */
short x=0,y=0,col;            /* x,y->Punktkoordinaten               */
float cre,cim,re,im,re1,re2,im1,im2,h1,h2,h3,h4;
                              /* cre,cim fuer Julia/Mandel           */
                              /* re,im laufender Re- und Im-Teil     */
                              /* re1,re2,im1,im2  -> Folgenglieder   */
                              /* h1,..h5 -> Hilfsvariable            */
float d[5];                   /* Abstaende zu den Nullstellen        */
char text[]="Komplexe Dynamische Systeme";
short x0=0,y0=0;              /* Koordinaten nach dem Laden          */
short width,height;           /* zum Rechnen                         */
short xs,ys,ye,fxoff,fyoff,yoff;
short bwf,britef,dreidf,pagef;/* Flags fuer Extra-Menu               */
int gridstep=10;              /* Anzahl der Gitterunterteilungen     */
int fak=1;                    /* Multiplikator fuer n bei 3d Ausgabe */


/************************/
/* Hauptprogramm       */
/************************/

main()
{
 emin*=emin,emax*=emax;
 open_things();
 for(;;)  /* solange Programm nicht beendet wird */
 {
    while((MyMessage=GetMsg(MyWindow->UserPort))==0);
    ReplyMsg(MyMessage);
    switch(MyMessage->Class)
    {
```

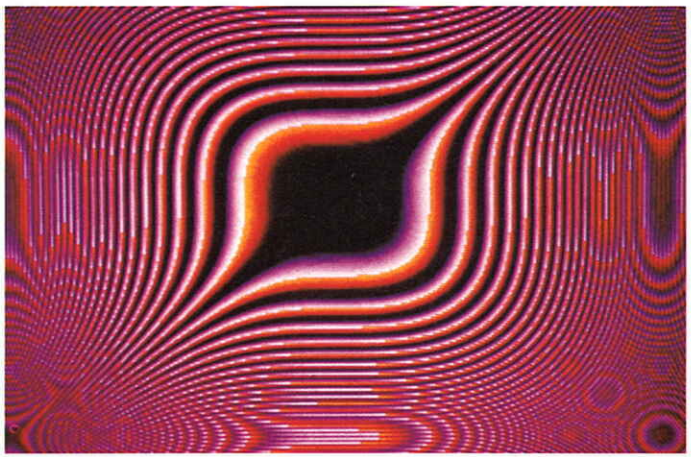
```
111         case CLOSEWINDOW: close_things();
112         case GADGETUP:    /* nur ein Gadget! */
113                           strcpy(File,F);
114                           for(i=Ø;File[i]!=' '&&i<8Ø;++i);
115                           File[i]=Ø;
116                           strcpy(Fileh,File);
117                           strcpy(Fileh+strlen(Fileh),".info");
118                           break;
119         case MENUPICK:    do_menu(MyMessage->Code);
12Ø                          break;
121       }
122   }
123 }
124
125
126 /***********************/
127 /* Initialisierung     */
128 /***********************/
129
13Ø open_things()
131 {
132  IntuitionBase=OpenLibrary("intuition.library",Ø);
133  if(IntuitionBase==Ø)
134     close_things();
135  GfxBase=OpenLibrary("graphics.library",Ø);
136  if(GfxBase==Ø)
137     close_things();
138  MyWindow=OpenWindow(&MyNewWindow);
139  if(MyWindow==Ø)
14Ø     close_things();
141  SetMenuStrip(MyWindow,&MYFIRSTMENU);
142  SetWindowTitles(MyWindow,MyNewWindow.Title,"von H. Knappe fÜr M&T 1987");
143 }
144
145
146 close_things()
147 {
148  if(IntuitionBase!=Ø) CloseLibrary(IntuitionBase);
149  if(GfxBase!=Ø) CloseLibrary(GfxBase);
15Ø  if(MyWindow!=Ø) CloseWindow(MyWindow);
151  exit();
152 }
153
154 /* Screen des gewaehlten Formats oeffnen */
155 open_work()
156 {
157  TOPEDGE=25;
158  switch(resolution)
159  {
16Ø     case Ø:
161       WIDTH=32Ø;HEIGHT=25Ø;DEPTH=5;FLAGS=Ø;anzcol=32;
162       fxoff=1Ø;fyoff=95;yoff=5;width=15Ø;height=15Ø;break;
163     case 1:
164       WIDTH=64Ø;HEIGHT=25Ø;DEPTH=4;FLAGS=HIRES;anzcol=16;
165       fxoff=2Ø;fyoff=95;yoff=5;width=45Ø;height=15Ø;break;
166     case 2:
167       WIDTH=32Ø;HEIGHT=5ØØ;DEPTH=5;FLAGS=LACE;anzcol=32;
168       fxoff=1Ø;fyoff=19Ø;yoff=1Ø;width=1ØØ;height=2ØØ;break;
169     case 3:
17Ø       WIDTH=64Ø;HEIGHT=5ØØ;DEPTH=4;FLAGS=HIRES+LACE;anzcol=8;
171       fxoff=2Ø;fyoff=19Ø;yoff=1Ø;width=3ØØ;height=3ØØ;break;
172     default: /* unsinnige Aufloesung */
173       DisplayBeep(Ø);
174       return(1);
175  }
176  if((resolution==2)(resolution==3))
177     TOPEDGE=5Ø;
178  if((DEPTH==5)&&(britef!=Ø))
179  {
18Ø     DEPTH=6;  /* Extra_HalfBrite Modus setzen */
181     anzcol=64;
182     if(resolution==Ø) FLAGS=EXTRA_HALFBRITE;
183     if(resolution==2) FLAGS=EXTRA_HALFBRITE+LACE;
184  }
185  MyScreen=OpenScreen(&MyNewScreen);
186  set_colors();
187  if(MyScreen==Ø)
188  {
189     DisplayBeep(Ø);
19Ø     return(1);
191  }
192  /* Oeffnen eines Screen fuellenden Windows */
193  NewWorkWindow.Screen=MyScreen;
194  WorkWindow=OpenWindow(&NewWorkWindow);
```







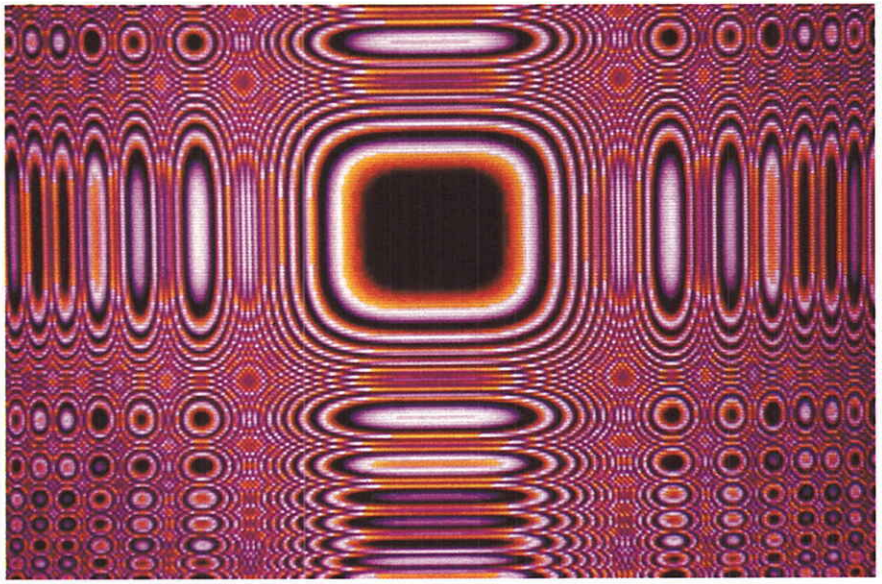


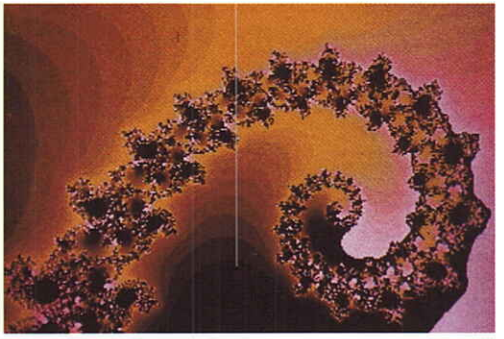


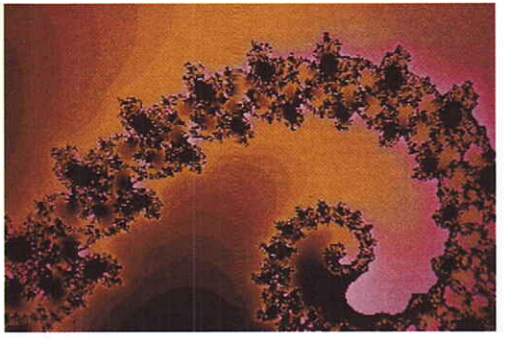
Bilder von links nach rechts:

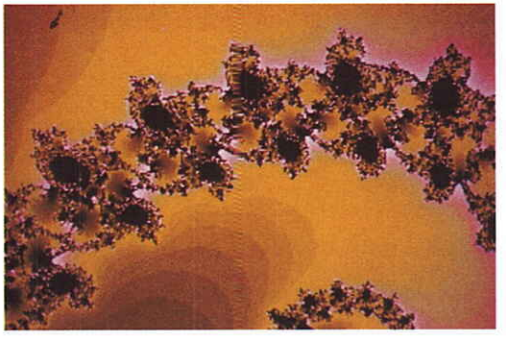
Tafel 1: Feigenbaum-Diagramm
Tafel 2: dreidimensionales Feigenbaum-Gebirge
Tafel 3: verschiedene Stadien einer Cosinus-Welle
Tafel 4 und Tafel 5: Tapetenmuster

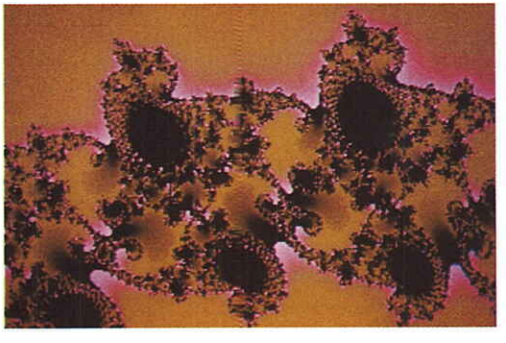
Bilder unten:
Tafel 6 und Tafel 7: Tapetenmuster



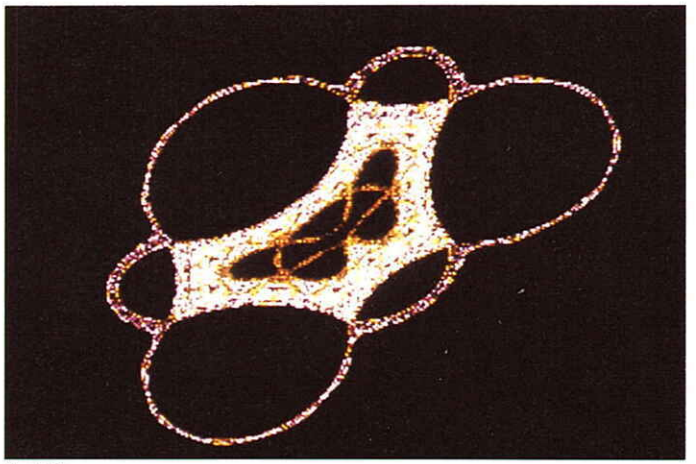








Bilder rechts:
Tafel 8(1)–8(4): Ein Flug ins Apfelmännchen

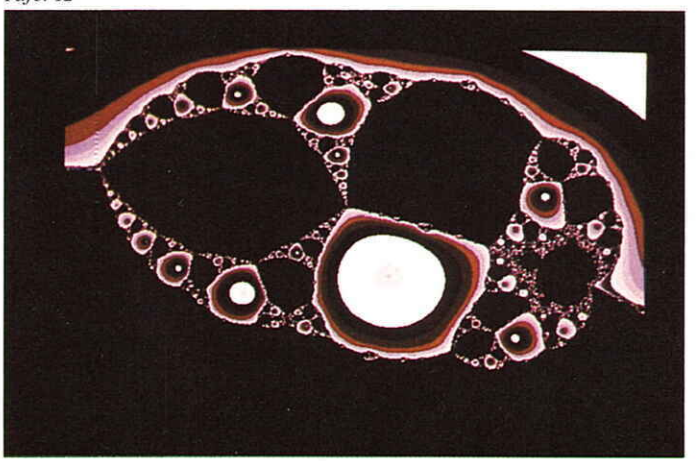


Tafel 9

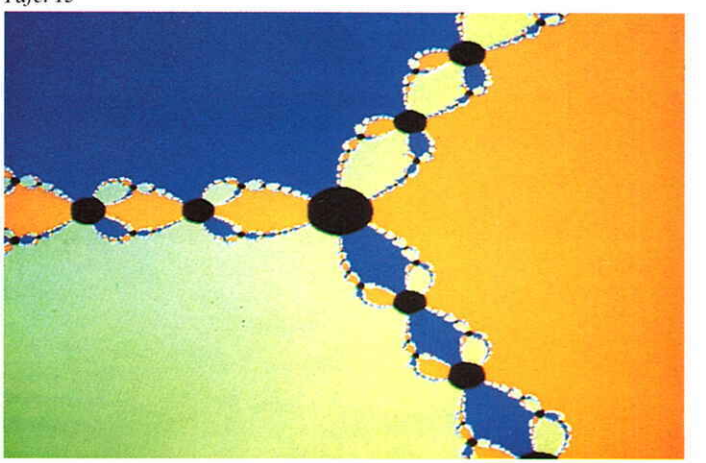
Tafel 10: Eine künstliche Landschaft

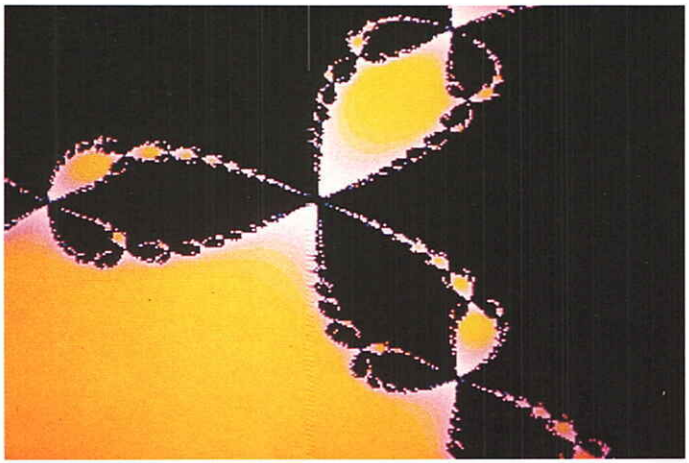
Tafel 11(1)–11(3): »Verzerrungen«, Vergrößerung von links nach rechts

Tafel 12

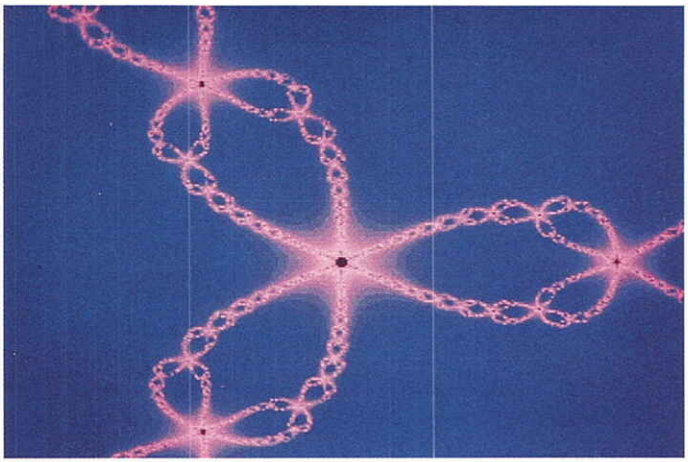


Tafel 13

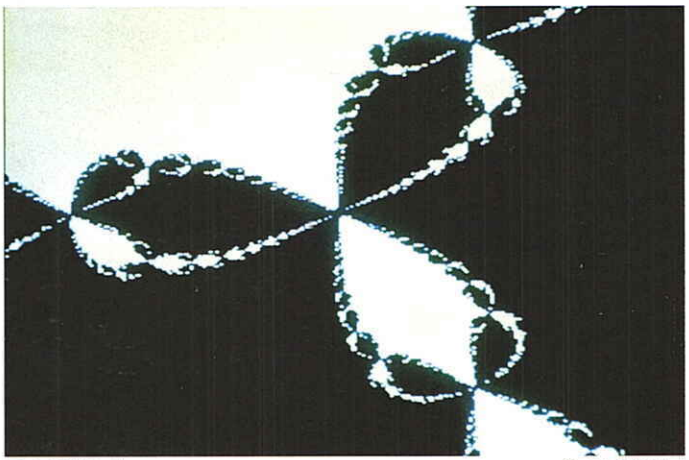




Tafel 14(1)

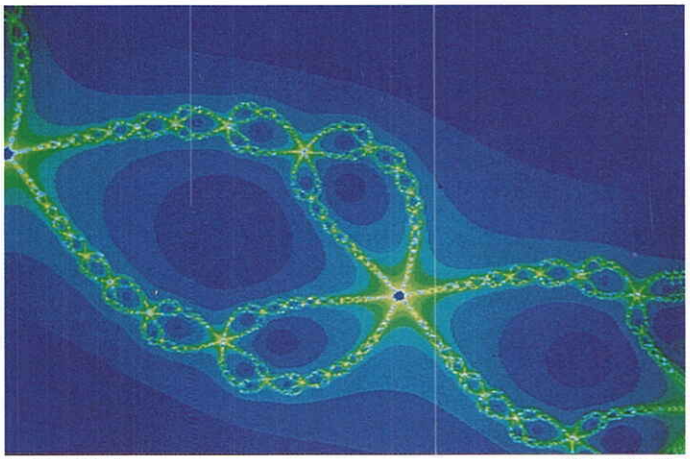


Tafel 15(1)



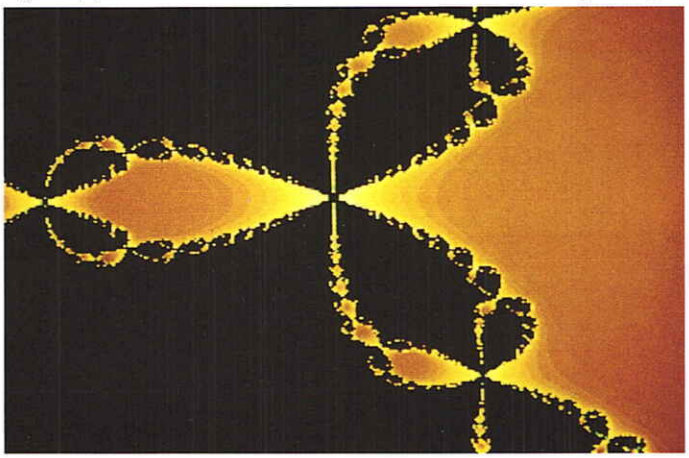
Tafel 14(2) ▲

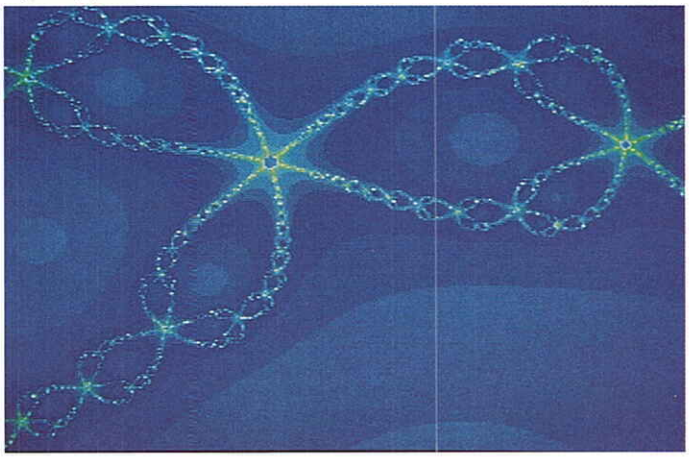
Tafel 14(3) ▼

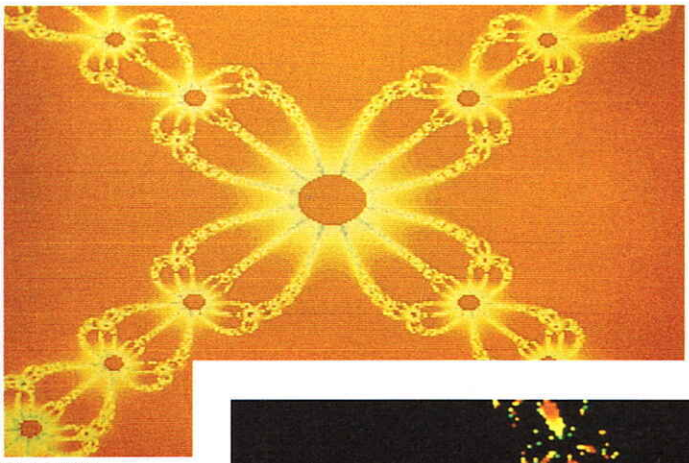


Tafel 15(2) ▲

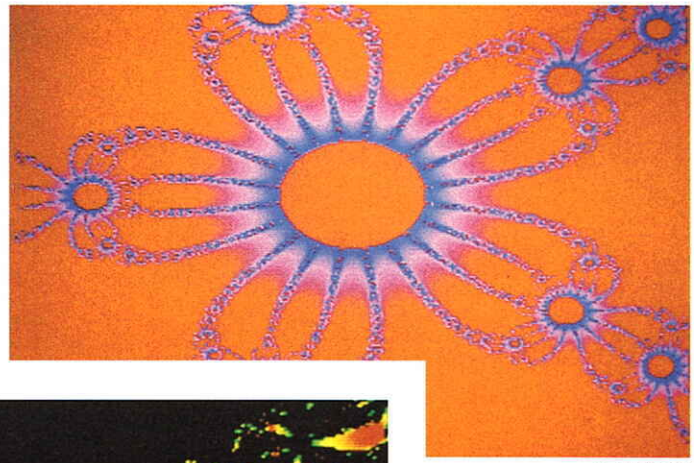
Tafel 15(3) ▼



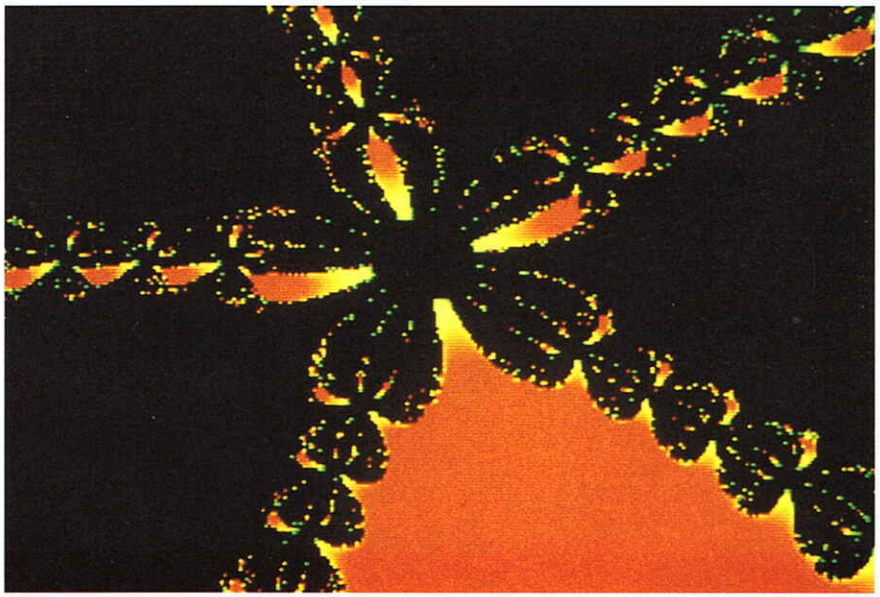




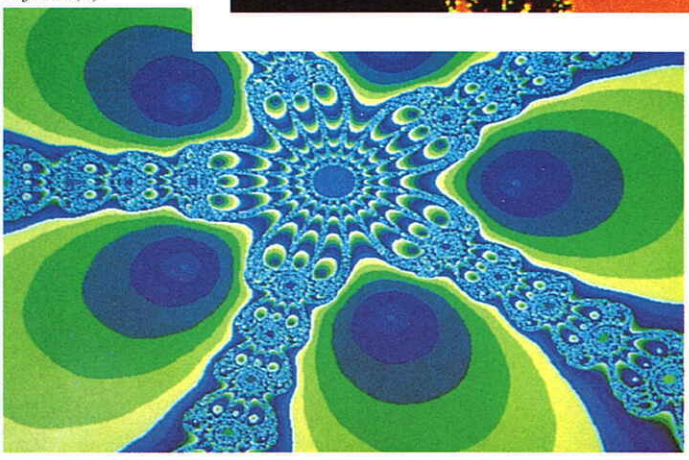
Tafel 16(1)



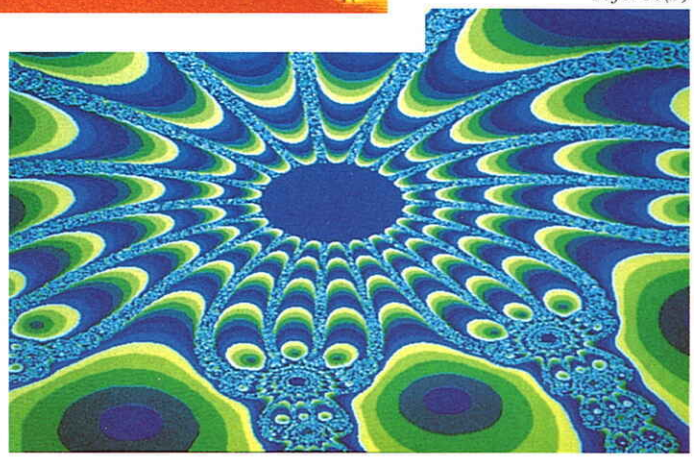
Tafel 16(2)



Tafel 16(3)

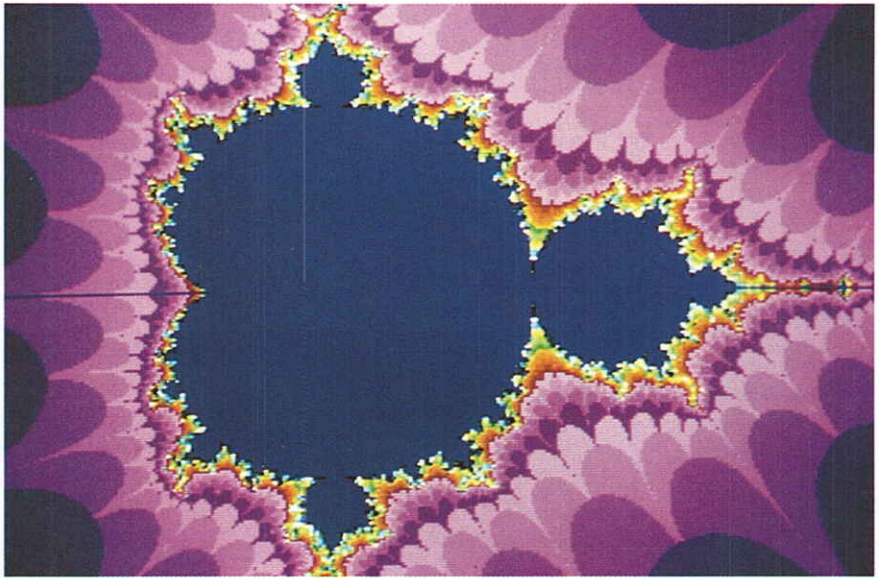


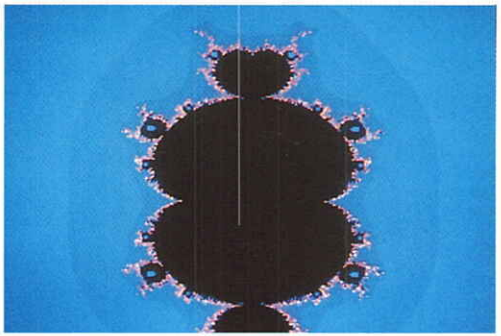
Tafel 16(4)

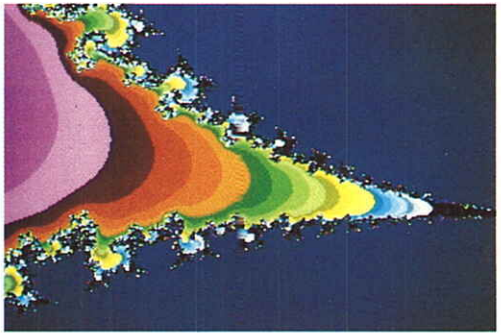


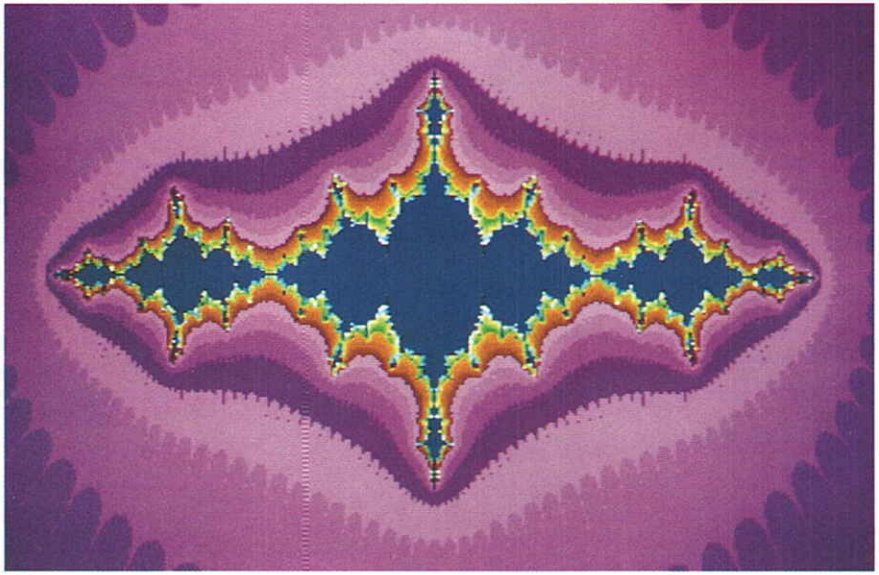
Tafel 16(5)

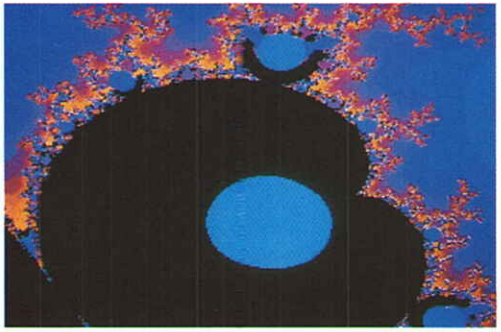
Tafel 17(1)–17(2):
Das Apfelmännchen



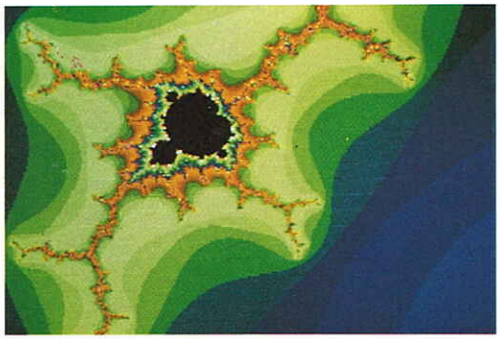


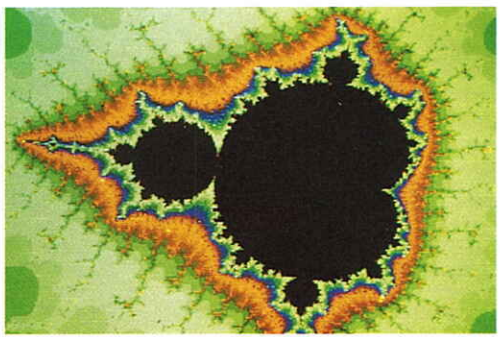


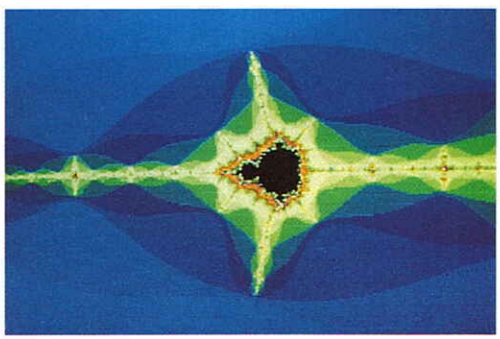


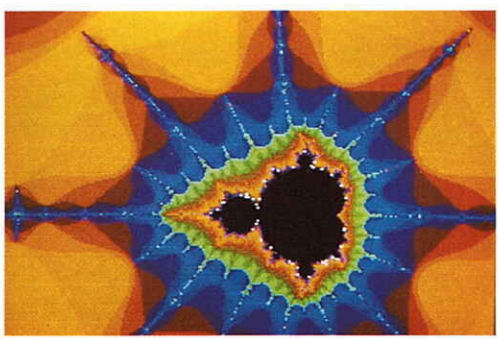


Bilder rechts:
Tafel 18(1)–18(4): Die Folge z^3-c/Auswertung in der c-Ebene

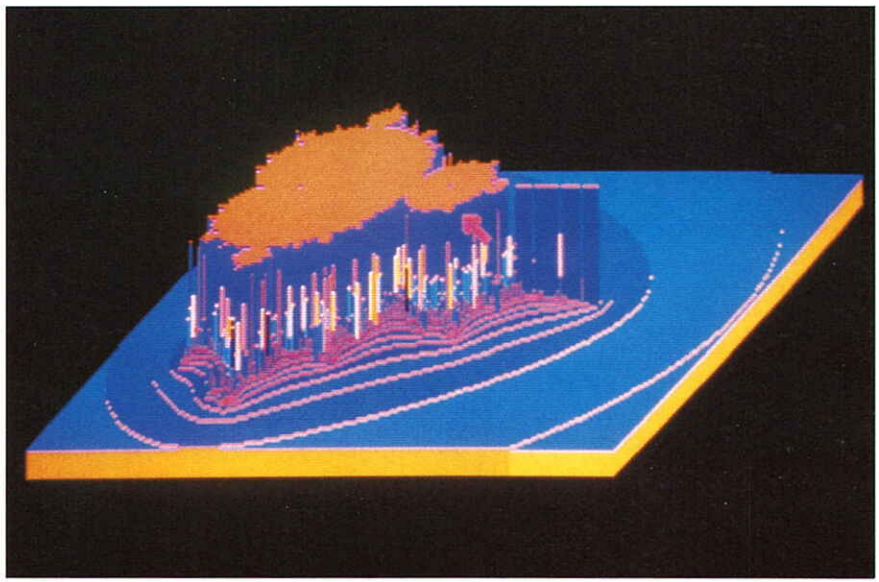


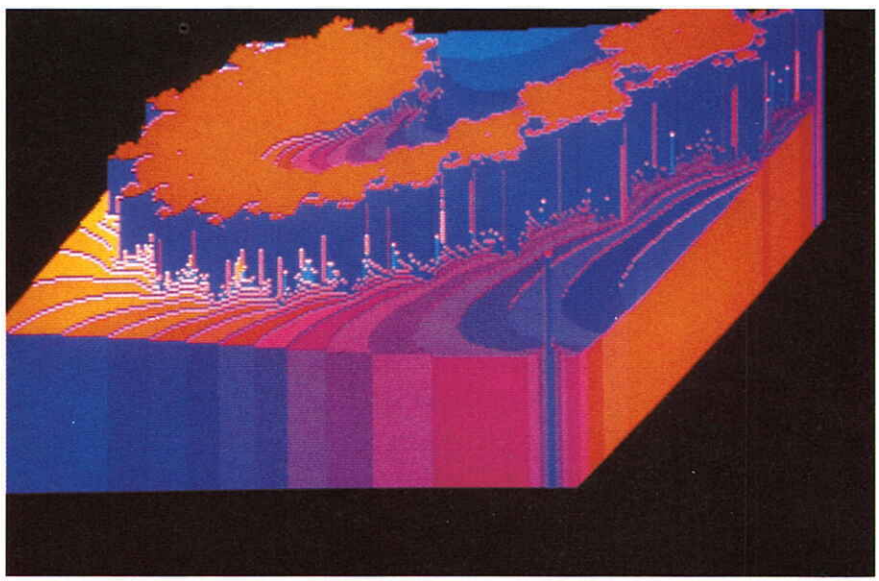




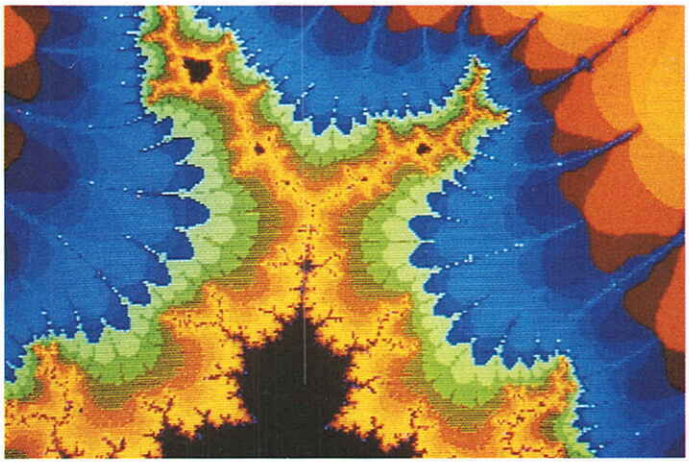


Bilder unten:
Tafel 19(1)–19(2):
Dreidimensionale Mandelbrot-Landschaften

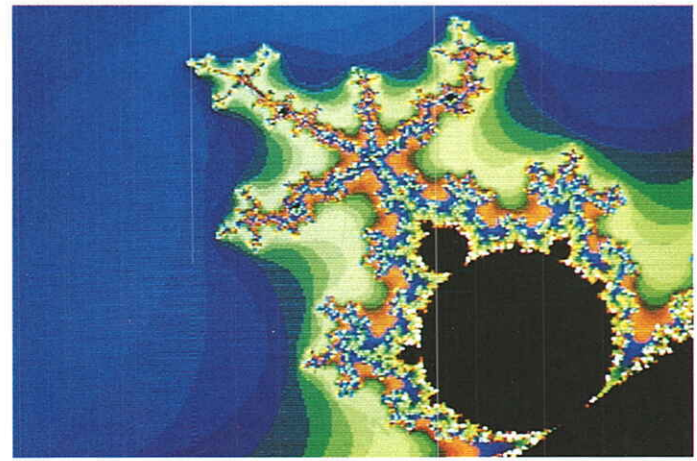




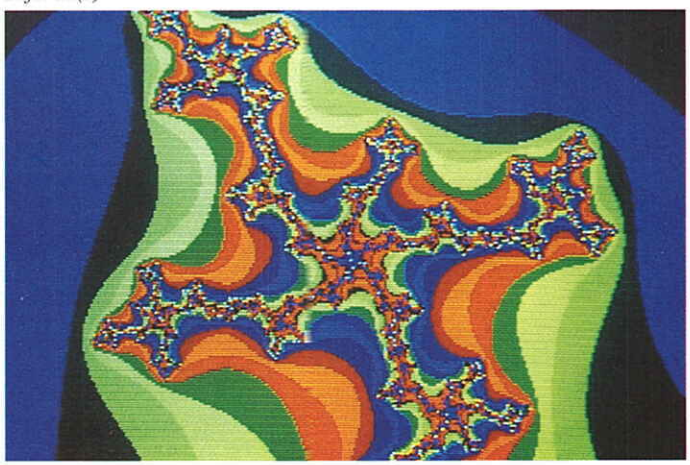
Bilder links:
Tafel 20(1)–20(4): Ausschnitte am Rand der Mandelbrot-Menge



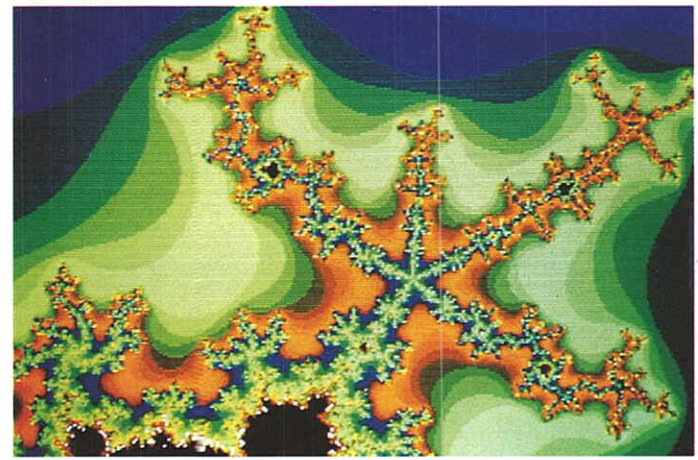
Tafel 21(1)



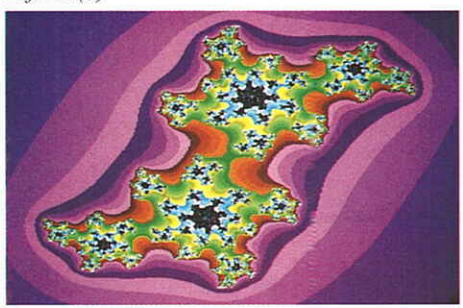
Tafel 21(2)

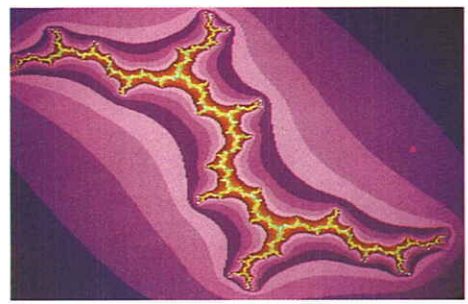


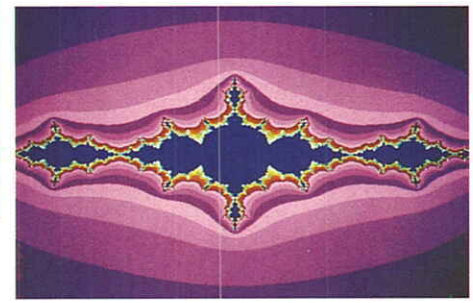
Tafel 21(3)



Tafel 21(4)

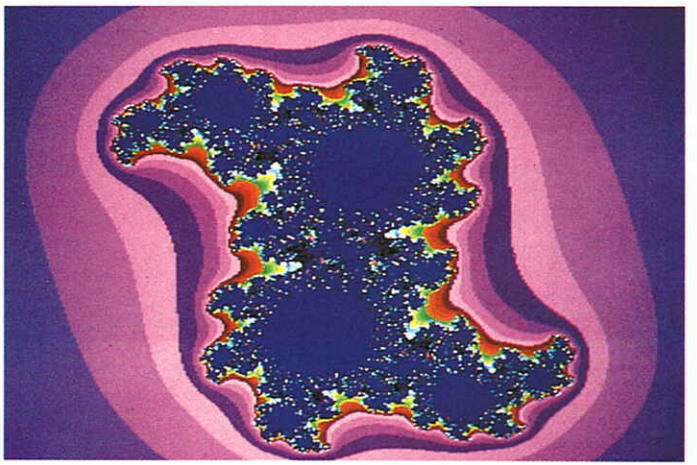


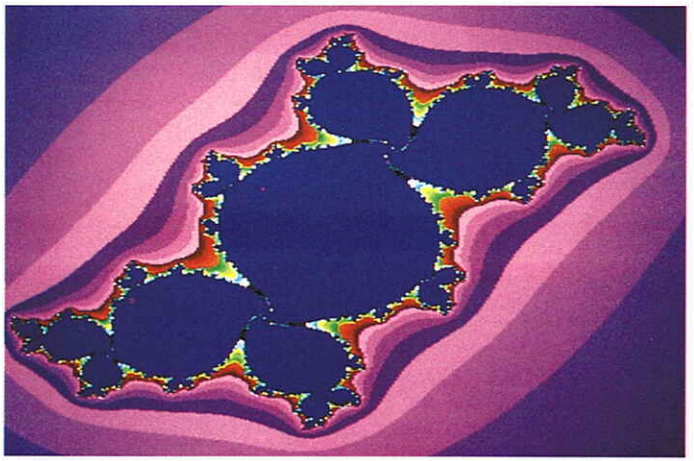


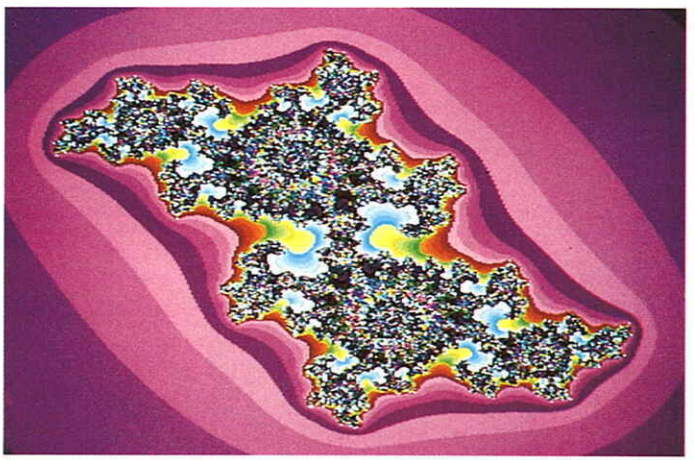


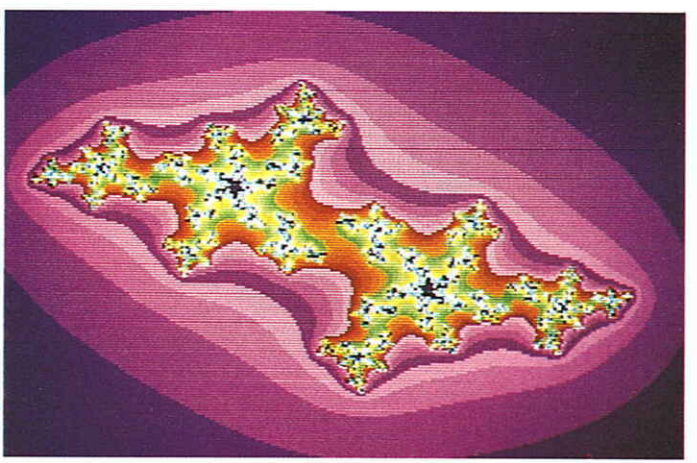
Tafel 22(1)–22(3):
Julia-Mengen (z^2-c)

Tafel 23(1)–23(4):
Julia-Mengen

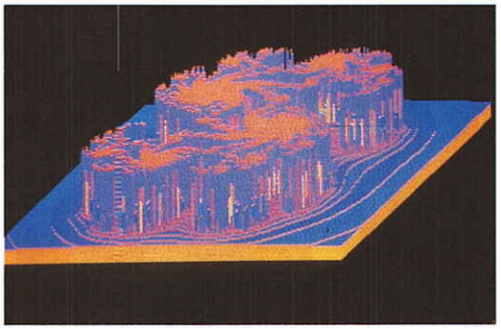


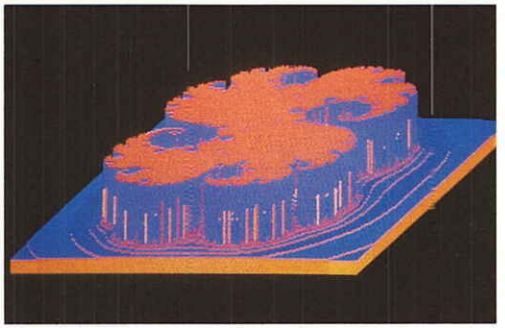




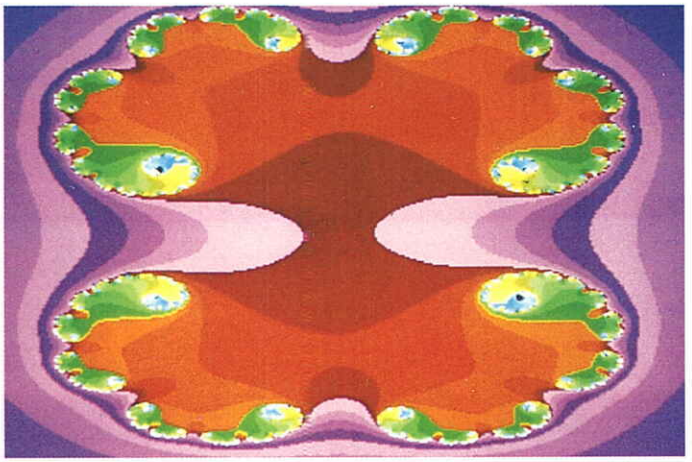


Tafel 24(1)



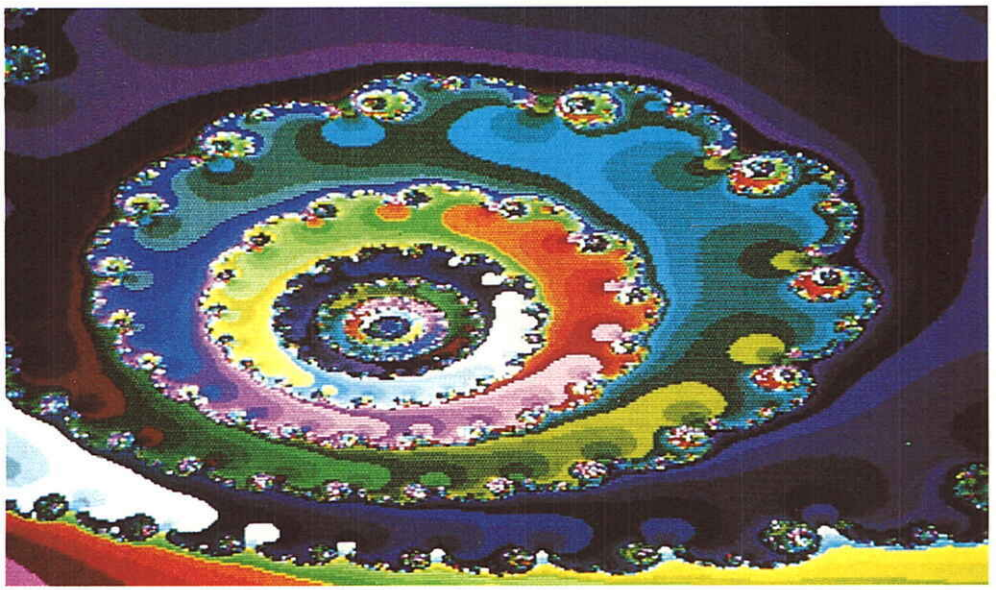


Tafel 24(2)

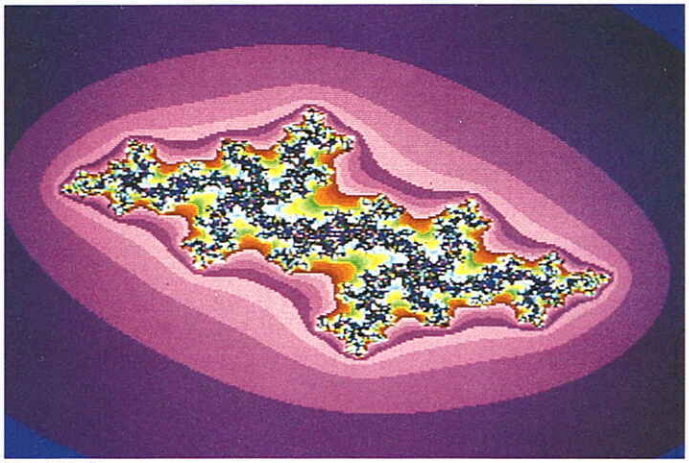


Tafel 25(1)

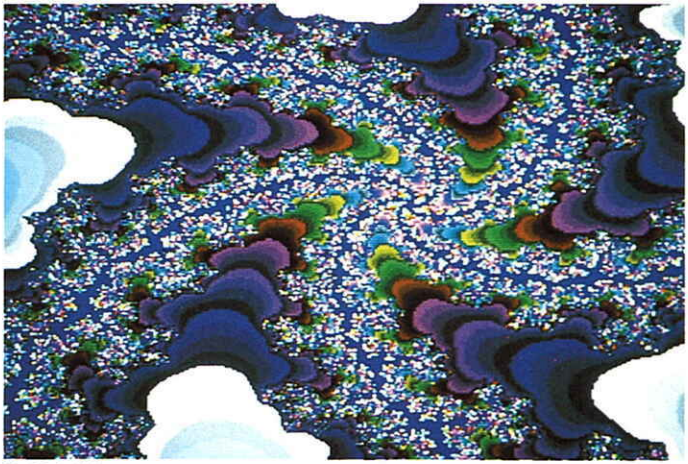
Tafel 25(2)



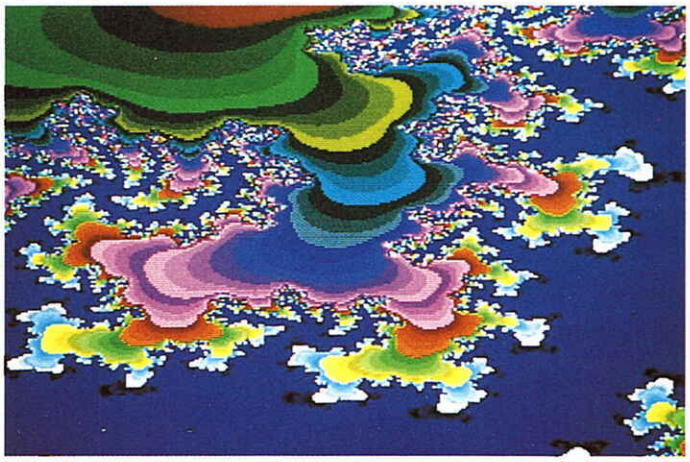
Tafel 25(3)



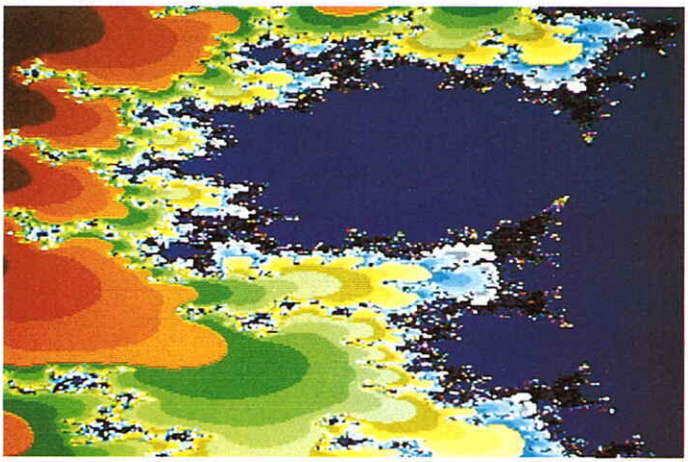
Tafel 26(1)



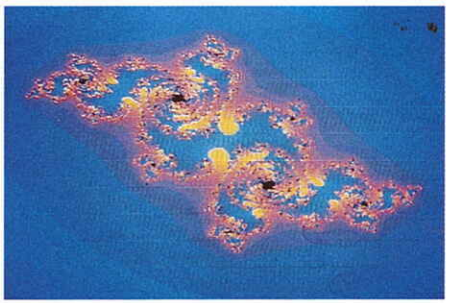
Tafel 26(2)



Tafel 26(3)

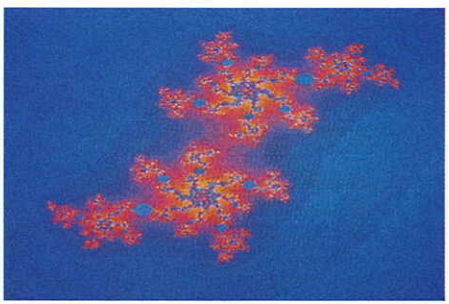


Tafel 26(4)

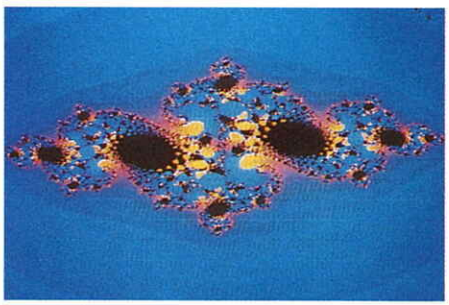


Tafel 27(1)

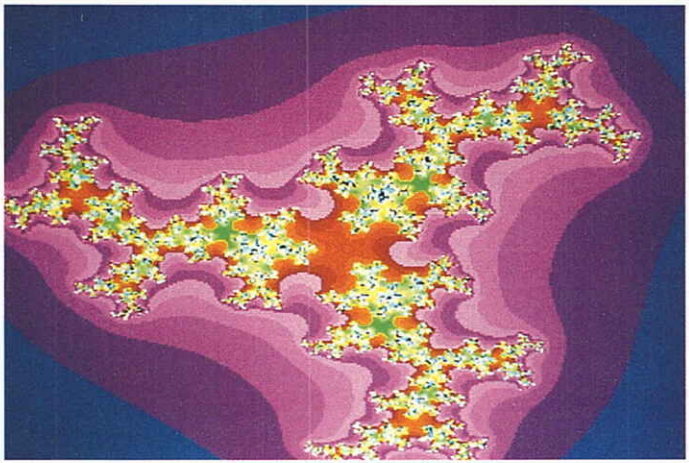
Julia-Mengen

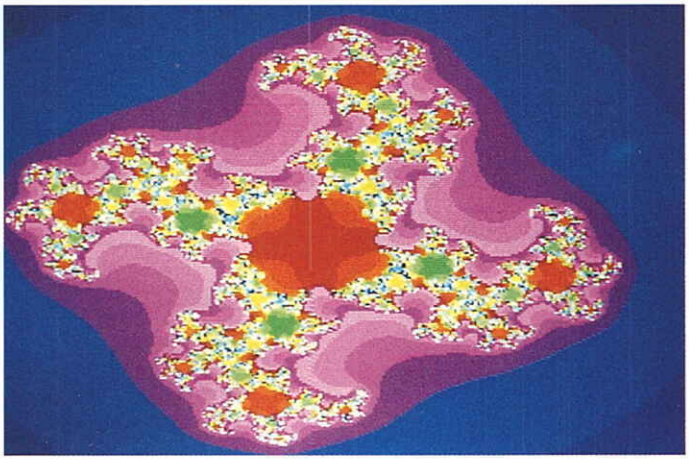


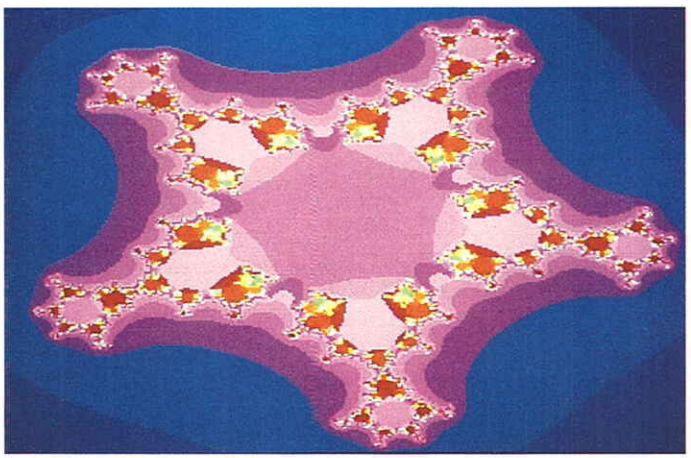
Tafel 27(2)



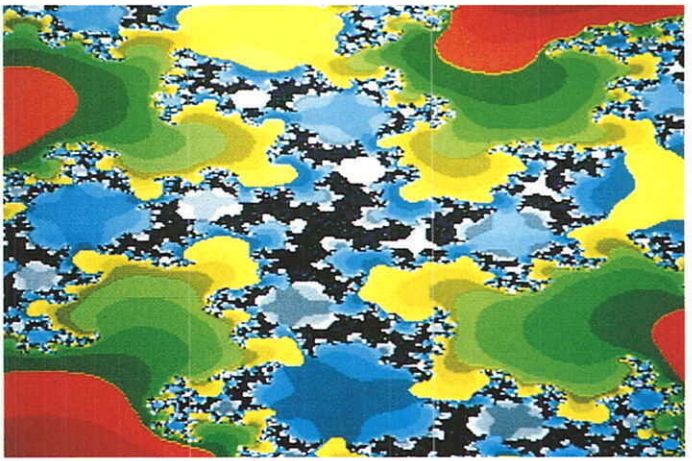
Tafel 27(3)

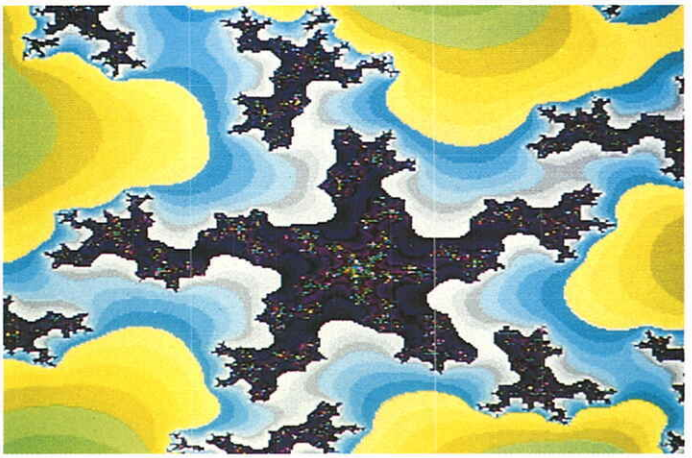






Bilder links:
Tafel 28(1) Julia-Mengen
28(2) höherer Ordnung
28(4)





Bilder oben:
Tafel 28(3)(1)–28(3)(2): Vergrößerung

```
195  if(WorkWindow==Ø)
196  {
197     DisplayBeep(Ø);
198     close_work();
199     return(1);
2ØØ  }
2Ø1  SetMenuStrip(WorkWindow,&WORKMENU);
2Ø2  rp=WorkWindow->RPort;
2Ø3  return(Ø);
2Ø4 }
2Ø5
2Ø6 close_work()
2Ø7 {
2Ø8  xØ=yØ=Ø;
2Ø9  BW=OFF,bwf=Ø;
21Ø  PAGE=OFF,pagef=Ø;
211  if(WorkWindow!=Ø) CloseWindow(WorkWindow);
212  if(MyScreen!=Ø) CloseScreen(MyScreen);
213  WorkWindow=MyScreen=Ø;
214 }
215
216
217 set_colors()
218 {
219  SHORT j;
22Ø  struct ViewPort *vp;
221
222  vp=&(MyScreen->ViewPort);
223  if((britef==Ø)(resolution==1)(resolution==3))
224  {
225  SetRGB4(vp,Ø,Ø,Ø,Ø);      /* Hintergrund schwarz */
226  for(i=1;i<anzcol/2;++i)
227     SetRGB4(vp,i,(int)(15*((float)i*2/(float)anzcol)),Ø,Ø);
228  for(i=anzcol/2;i<anzcol;++i)
229     SetRGB4(vp,i,15-(int)(15*((float)i/(float)anzcol)),Ø,
23Ø           (int)(15*((float)i/(float)anzcol)));
231  }
232  else
233  { /* fuer Brite Modus Farben setzen */
234     for(i=8,j=Ø;i<16;i+=2,++j)
235     {
236        SetRGB4(vp,j,Ø,Ø,i);
237        SetRGB4(vp,j+4,i-1,Ø,i);
238        SetRGB4(vp,j+8,i,Ø,i);
239        SetRGB4(vp,j+12,i,Ø,Ø);
24Ø        SetRGB4(vp,j+16,Ø,i,Ø);
241        SetRGB4(vp,j+2Ø,i,i,Ø);
242        SetRGB4(vp,j+24,Ø,i,i);
243        SetRGB4(vp,j+28,i,i,i);
244     }
245  }
246 }
247
248 set_bw()
249 {
25Ø  struct ViewPort *vp;
251
252  vp=&(MyScreen->ViewPort);
253  for(i=Ø;i<anzcol;i+=2) SetRGB4(vp,i,Ø,Ø,Ø);    /* schwarz */
254  for(i=1;i<anzcol;i+=2) SetRGB4(vp,i,15,15,15); /* weiss */
255 }
256
257
258 /***********************/
259 /* Menu-Abfrage        */
26Ø /***********************/
261
262 do_menu(MyCode)
263 USHORT MyCode;
264 {
265  short menu=MENUNUM(MyCode),item=ITEMNUM(MyCode),subitem=SUBNUM(MyCode);
266  long file;
267
268  switch(menu)
269  {
27Ø     case Ø: /* Projekt Menu */
271     switch(item)
272     {
273        case Ø: /* Laden */
274           if((file=Open(Fileh,MODE_OLDFILE))==Ø)
275           {
276              if(load(File,Ø)!=Ø)
277              {
```

```
278             DisplayBeep(Ø);  /* erfolgreich? */
279             break;
28Ø           }
281         }
282         else
283         {
284           Read(file,&MyInfo,sizeof(MyInfo));
285           Close(file);
286           rmin=MyInfo.rmin,rmax=MyInfo.rmax;
287           imin=MyInfo.imin,imax=MyInfo.imax;
288           cim=MyInfo.cim,cre=MyInfo.cre;
289           xØ=MyInfo.mx,yØ=MyInfo.my;
29Ø           emin=MyInfo.emin;   /* Quadrate abgespeichert! */
291           emax=MyInfo.emax,nmax=MyInfo.nmax,noff=MyInfo.noff;
292           funktion=MyInfo.mode,resolution=MyInfo.res;
293           dreidf=MyInfo.dim;  /* 3d Modus? */
294           britef=MyInfo.half_brite; /* HALF_BRITE? */
295           if(MyInfo.mode<ØMyInfo.mode>ANZFUNK-1)
296           {
297             DisplayBeep(Ø);
298             break; break;
299           }
3ØØ           set_menu();
3Ø1           if(open_work()!=Ø) /* Konnte Work-Window geoeffnet werden? */
3Ø2           {
3Ø3             DisplayBeep(Ø);
3Ø4             break; break;
3Ø5           }
3Ø6           else
3Ø7           {
3Ø8             ClearMenuStrip(MyWindow);
3Ø9             if(load(File,MyScreen)!=Ø)
31Ø               DisplayBeep(Ø);
311             else
312             {
313               set_colors();
314               fractal();
315             }
316             close_work();
317             SetMenuStrip(MyWindow,&MYFIRSTMENU);
318           }
319         }
32Ø         break;
321       case 1: /* Start */
322         if(open_work()!=Ø) /* Konnte Work-Window geoeffnet werden? */
323           break;
324         else
325         {
326           ClearMenuStrip(MyWindow);
327           fractal();
328           close_work();
329           SetMenuStrip(MyWindow,&MYFIRSTMENU);
33Ø         }
331         break;
332       case 2: /* Input */
333         inpcon();
334         break;
335       case 3: /* Ende */
336         close_things();
337     }
338     break;
339     case 1: /* Format */
34Ø       resolution=item;
341       set_resolution(resolution);
342       break;
343     case 2: /* Funktion */
344       funktion=item;
345       set_funktion(funktion);
346       break;
347     case 3: /* Extra */
348     switch(item)
349     {
35Ø       case Ø: /* Brite */
351         BRITE=OFF;
352         if(britef==Ø)
353         {
354           BRITE=ON;
355           britef=1;
356         }
357         else
358           britef=Ø;
359         break;
36Ø       case 1: /* 3-D */
```

```
361          switch(subitem)
362          {
363             case Ø: /* normal */
364                DREIDN=DREIDI=OFF;
365                if(dreidf==Ødreidf==2)
366                {
367                   DREIDN=ON;
368                   dreidf=1;
369                }
37Ø                else
371                   dreidf=Ø;
372                break;
373             case 1: /* invertiert */
374                DREIDN=DREIDI=OFF;
375                if(dreidf==Ødreidf==1)
376                {
377                   DREIDI=ON;
378                   dreidf=2;
379                }
38Ø                else
381                   dreidf=Ø;
382                break;
383          }
384          break;
385       }
386       break;
387       default: /* nichts ist eingetreten! */
388       break;
389    }
39Ø }
391
392 set_menu()
393 {
394  set_resolution(resolution);
395  set_funktion(funktion);
396  set_brite(britef);
397  set_3d(dreidf);
398 }
399
```

```
4ØØ set_resolution(i)
4Ø1 short i;
4Ø2 {
4Ø3  switch(i)
4Ø4  {
4Ø5     case Ø: /* (32Øx25Ø) 5 Planes */
4Ø6        LORES=ON; MEDRES=HILORES=HIMEDRES=OFF;
4Ø7        break;
4Ø8     case 1: /* (64Øx25Ø) 4 Planes */
4Ø9        MEDRES=ON; LORES=HILORES=HIMEDRES=OFF;
41Ø        break;
411     case 2: /* (32Øx5ØØ) 5 Planes */
412        HILORES=ON; LORES=MEDRES=HIMEDRES=OFF;
413        break;
414     case 3: /* (64Øx5ØØ) 4 Planes */
415        HIMEDRES=ON; LORES=MEDRES=HILORES=OFF;
416        break;
417  }
418 }
419
42Ø set_funktion(i)
421 short i;
422 {
423  switch(i)
424  {
425     case Ø: /* z^3-1=Ø */
426        F1=ON; F2=F3=F4=F5=F6=F7=F8=F9=F1Ø=F11=F12=OFF;
427        break;
428     case 1: /* z^4-1=Ø */
429        F2=ON; F1=F3=F4=F5=F6=F7=F8=F9=F1Ø=F11=F12=OFF;
43Ø        break;
431     case 2: /* z^5-1=Ø */
432        F3=ON; F1=F2=F4=F5=F6=F7=F8=F9=F1Ø=F11=F12=OFF;
433        break;
434     case 3: /* z^2-c=z */
435        F4=ON; F1=F2=F3=F5=F6=F7=F8=F9=F1Ø=F11=F12=OFF;
436        break;
437     case 4: /* z^3-c=z */
438        F5=ON; F1=F2=F3=F4=F6=F7=F8=F9=F1Ø=F11=F12=OFF;
439        break;
```

```
44Ø     case 5: /* z^4-c=z */
441        F6=ON; F1=F2=F3=F4=F5=F7=F8=F9=F1Ø=F11=F12=OFF;
442        break;
443     case 6: /* z^5-c=z */
444        F7=ON; F1=F2=F3=F4=F5=F6=F8=F9=F1Ø=F11=F12=OFF;
445        break;
446     case 7: /* Julia */
447        F8=ON; F1=F2=F3=F4=F5=F6=F7=F9=F1Ø=F11=F12=OFF;
448        break;
449     case 8: /* 'Julia' z^3-c */
45Ø        F9=ON; F1=F2=F3=F4=F5=F6=F7=F8=F1Ø=F11=F12=OFF;
451        break;
452     case 9: /* 'Julia' z^4-c */
453        F1Ø=ON; F1=F2=F3=F4=F5=F6=F7=F8=F9=F11=F12=OFF;
454        break;
455     case 1Ø: /* 'Julia' z^5-c */
456        F11=ON; F1=F2=F3=F4=F5=F6=F7=F8=F9=F1Ø=F12=OFF;
457        break;
458     case 11:
459        /* 'Erw.' Erweiterung zum einfachen Einbau einer eigenen Fkt. */
46Ø        F12=ON; F1=F2=F3=F4=F5=F6=F7=F8=F9=F1Ø=F11=OFF;
461        break;
462  }
463 }
464
465 set_brite(i)
466 short i;
467 {
468  BRITE=OFF;
469  if(britef==1)
47Ø     BRITE=ON;
471 }
472
473 set_3d(i)
474 short i;
475 {
476  DREIDN=DREIDI=OFF;
477  switch(i)
478  {
479     case 1: DREIDN=ON;
48Ø           break;
481     case 2: DREIDI=ON;
482           break;
483  }
484 }
485
486
487 /**********************/
488 /* Work-Window        */
489 /* bearbeiten         */
49Ø /**********************/
491
492 /* Welches Menue wurde im Work-Window angwewaehlt? */
493 /* Routine kehrt mit Werten von 1 bis 3 zurueck.   */
494 /* Ø -> kein Item angewaehlt!                      */
495 /* 1 -> Bild Speichern!                            */
496 /* 2 -> Work Fenster beenden!                      */
497 /* 3 -> Mousebutton gedrueckt!                     */
498 whats_up(class,code)
499 ULONG class;
5ØØ USHORT code;
5Ø1 {
5Ø2  USHORT menu,item;
5Ø3  long file;
5Ø4
5Ø5  switch(class)
5Ø6  {
5Ø7     case MENUPICK:       /* Menu wurde angewaehlt */
5Ø8                  menu=MENUNUM(code);
5Ø9                  item=ITEMNUM(code);
51Ø                  switch(menu)
511                  {
512                  case Ø: /* Projekt-Menu? */
513                     switch(item)
514                     {
515                        case Ø: /* Speichern?    */
516                        /* Speichern des Bildes im IFF-Format (kompr.) */
517                        /* mit zuletzt eingegebener Name */
518                        strcpy(File,F);
519                        for(i=Ø;File[i]!=' '&&i<8Ø;++i);
52Ø                        File[i]=Ø;
521                        strcpy(Fileh,File);
522                        strcpy(Fileh+strlen(Fileh),".info");
```

```
523                MyInfo.rmin=rmin,MyInfo.rmax=rmax;
524                MyInfo.imin=imin,MyInfo.imax=imax;
525                MyInfo.cim=cim,MyInfo.cre=cre;
526                MyInfo.mx=x,MyInfo.my=y;
527                MyInfo.emin=emin,MyInfo.emax=emax;
528                MyInfo.nmax=nmax,MyInfo.noff;
529                MyInfo.mode=funktion,MyInfo.res=resolution;
530                MyInfo.dim=dreidf; /* 3d Modus? */
531                MyInfo.half_brite=britef; /* HALF_BRITE? */
532                strcpy(MyInfo.text,text);
533                if(save(File,1,MyScreen)==Ø);
534                /* Bild kann nicht gespeichert werden!! */
535                /* Speichern des '.info' Files */
536                if((file=Open(Fileh,MODE_NEWFILE))!=Ø)
537                {
538                   Write(file,&MyInfo,sizeof(MyInfo));
539                   Close(file);
540                }
541                return(1);
542                   break;
543                case 1: /* Cancel?       */
544                   return(2);
545                   break;
546             }
547             break;
548          case 1: /* Extra-Menu */
549             switch(item)
550             {
551                case Ø: /* ganzer Screen? */
552                   PAGE=OFF;
553                   if(pagef==Ø)
554                   {
555                      pagef=1; PAGE=ON;
556                      while(MyScreen->TopEdge>Ø)
557                         MoveScreen(MyScreen,Ø,-1);
558                   }
559                   else
560                   {
561                      pagef=Ø;
562                      while(MyScreen->TopEdge<TOPEDGE)
563                         MoveScreen(MyScreen,Ø,1);
564                   }
565                      break;
566                   case 1: /* Schwarz&Weiss an/aus */
567                      BW=OFF;
568                      if(bwf==Ø)
569                      {
570                         BW=ON;
571                         bwf=1;
572                         set_bw(); /* Schwarz-Weiss setzen */
573                      }
574                      else
575                      {
576                         set_colors(); /* Farben setzen */
577                         bwf=Ø;
578                      }
579                      break;
580                   case 2: /* Grid on/off */
581                      if(dreidf==Ø)
582                         grid(); /* Gitter zeichnen und warten */
583                      break;
584                   case 3: /* ViewPort einzeichnen */
585                      if(dreidf==Ø)
586                         draw_view();
587                      break;
588                }
589                break;
590             }
591       case MOUSEBUTTONS:   /* Taste wurde gedrueckt */
592                   return(3);
593                   break;
594       default: /* ???? */
595                   break;
596    }
597    return(Ø);
598 }
599
600 /* Eingabe von Werten ueber ein Consol-Fenster */
601 inpcon()
602 {
603    int confile;
604    char line[80];
605
```

```
confile=Open("con:0/0/320/80/Eingabe-Fenster",MODE_OLDFILE);
if(confile==0)
{
   DisplayBeep(0);
   close_();
}
while(inline(line,confile,80)!=0)
{
   switch(line[0])
   {
      case 10 : /* Return-Zeile */
         break;
      case 'd': /* default-Werte seten */
         cim=cre=0,nmax=100,noff=0,rmin=imin=-1,rmax=imax=1,fak=1;
         emin=.001*.001,emax=1000*1000,gridstep=10,vrmin=vrmax=vimin=vimax=0;
         break;
      case 'f': /* Faktor fuer 3D-Ausgabe */
         fak=1;
         if(sscanf(line+1,"%d",&fak)!=1)
            printf("Eingabefehler:  -> %s",line);
         break;
      case 'c': /* cre,cim einlesen */
         cim=cre=0;
         if(sscanf(line+1,"%f,%f",&cre,&cim)!=2)
            printf("Eingabefehler:  -> %s",line);
         break;
      case 'n': /* nmax,noff einlesen */
         nmax=100,noff=0;
         if(sscanf(line+1,"%d,%d",&nmax,&noff)!=2)
            printf("Eingabefehler: -> %s",line);
         break;
      case 'r': /* rmin,rmax einlesen */
         rmin=-1,rmax=1;
         if(sscanf(line+1,"%f,%f",&rmin,&rmax)!=2)
            printf("Eingabefehler: -> %s",line);
         break;
      case 'i': /* imin,imax einlesen */
         imin=-1,imax=1;
         if(sscanf(line+1,"%f,%f",&imin,&imax)!=2)
            printf("Eingabefehler: -> %s",line);
         break;
      case 'e': /* emax,emin einlesen */
         emin=.001,emax=1000;
         if(sscanf(line+1,"%f,%f",&emin,&emax)!=2)
            printf("Eingabefehler: -> %s",line);
         emin*=emin,emax*=emax;
         break;
      case 'v': /* 'ViewPort' */
         vrmin=vrmax=vimin=vimax=0;
         if(sscanf(line+1,"%f,%f,%f,%f",&vrmin,&vrmax,&vimin,&vimax)!=4)
            printf("Eingabefehler: -> %s",line);
         if(vrmin>vrmaxvimin>vimax)
         {
            printf("Eingabefehler: -> %s",line);
            vrmin=vrmax=vimin=vimax=0;
         }
         break;
      case 'g':
         gridstep=10;
         if(sscanf(line+1,"%d",&gridstep)!=1)
            printf("Eingabefehler: -> %s",line);
         break;
      case 'a': /* Ausgabe der Parameter */
         ausgabe();
         break;
      case 'x': /* Ende der Eingabe */
         Close(confile);
         ausgabe();
         return();
      default : /* Fehler aufgetreten */
         printf("Eingabefehler: -> %s",line);
         break;
   }
}

inline(c,handle,len) /* eine Zeile des Files einlesen */
char c[];
long *handle;
short len;
{
 short i=0,ende=0;

```

```
689  while((i<len)&&(c[i-1]!=1Ø)&&((ende=Read(handle,&c[i],1))!=Ø))
69Ø     ++i;
691  c[i]=Ø;
692  return(ende);
693  }
694
695  /* Ausgabe der aktuellen Parameter */
696  ausgabe()
697  {
698  printf("\nParameter:\n nmax=%d noff=%d\n emin=%f emax=%f\n",
699         nmax,noff,sqrt(emin),sqrt(emax));
7ØØ  printf(" cre=%f cim=%f\n",cre,cim);
7Ø1  printf(" Bereich [%f,%f] [%f,%f]\n",rmin,rmax,imin,imax);
7Ø2  printf(" View [%f,%f] [%f,%f]\n",vrmin,vrmax,vimin,vimax);
7Ø3  printf(" Gitter: %d\n",gridstep);
7Ø4  printf(" 3D-Faktor: %d\n",fak);
7Ø5  }
7Ø6
7Ø7
7Ø8  /* Gitter im COMPLEMENT-Modus zeichnen */
7Ø9  grid()
71Ø  {
711  UBYTE *press,j;
712
713  press=12574721;
714  SetDrMd(rp,COMPLEMENT);
715  for(j=Ø;j<2;++j)
716  {
717     for(i=1;i<gridstep;++i)
718     {
719        Move(rp,(short)(i*(float)width/(float)gridstep),Ø);
72Ø        Draw(rp,(short)(i*(float)width/(float)gridstep),height);
721        Move(rp,Ø,(short)(i*(float)height/(float)gridstep));
722        Draw(rp,width,(short)(i*(float)height/(float)gridstep));
723     }
724     if(j==Ø) while((*press&64)==64); /* nur ein Mal!!! */
725     /* warten bis linker Mousebutton gedrueckt wird */
726  }
727  SetDrMd(rp,JAM1);
728  }
729
```

```
73Ø  /* 'ViewFenster' im COMPLEMENT-Modus zeichnen */
731  draw_view()
732  {
733   short x1view,y1view,x2view,y2view;
734   UBYTE *press;
735
736   press=12574721;
737   x1view=(short)((vrmin-rmin)/rst),y1view=(short)((vimin-imin)/ist);
738   x2view=(short)((vrmax-rmin)/rst),y2view=(short)((vimax-imin)/ist);
739   if(x1view<Øx1view>widthy1view<Øy1view>height
74Ø      x2view<Øx2view>widthy2view<Øy2view>height)
741      return(FALSE);
742   SetDrMd(rp,COMPLEMENT);
743   RectFill(rp,x1view,y1view,x2view,y2view);
744   while((*press&64)==64); /* warten bis linker Mousebutton gedrueckt wird */
745   RectFill(rp,x1view,y1view,x2view,y2view);
746   SetDrMd(rp,JAM1);
747   return(TRUE);
748  }
749
75Ø
751  /***********************/
752  /* Berechnungsroutine  */
753  /* fractal()           */
754  /***********************/
755
756  fractal()
757  {
758   if(dreidf==Ø)
759   {
76Ø      width=WIDTH;
761      height=HEIGHT;
762   }
763   /* Definition der Bezugspunkte je nach Funktion */
764   switch(funktion)
765   {
766      case Ø: /* Newton 3 */
767         nsre[Ø]=1,nsim[Ø]=Ø;              /* 1.Nullstelle  */
768         nsre[1]=-.5,nsim[1]=.866;         /* 2.Nullstelle  */
769         nsre[2]=-.5,nsim[2]=-.866;        /* 3.Nullstelle  */
77Ø         break;
```

```
771     case 1: /* Newton 4 */
772        nsre[Ø]=1,nsim[Ø]=Ø;              /* 1.Nullstelle  */
773        nsre[1]=Ø,nsim[1]=1;              /* 2.Nullstelle  */
774        nsre[2]=-1,nsim[2]=Ø;             /* 3.Nullstelle  */
775        nsre[3]=Ø,nsim[3]=-1;             /* 4.Nullstelle  */
776        break;
777     case 2: /* Newton 5 */
778        nsre[Ø]=.3Ø9Ø,nsim[Ø]=.951Ø;      /* 1.Nullstelle */
779        nsre[1]=-.8Ø9Ø,nsim[1]=.5877;     /* 2.Nullstelle */
78Ø        nsre[2]=-.8Ø9Ø,nsim[2]=-.5877;    /* 3.Nullstelle */
781        nsre[3]=.3Ø9Ø,nsim[3]=-.951Ø;     /* 4.Nullstelle */
782        nsre[4]=1,nsim[4]=Ø;              /* 5.Nullstelle */
783        break;
784     default: /* sonst Konvergenzuntersuchung gegen (Ø,Ø) */
785        nsre[Ø]=nsim[Ø]=Ø;
786        break;
787  }
788  if(xØ==width) xØ=Ø;
789  if(yØ==height) yØ=Ø;
79Ø  rst=(rmax-rmin)/width,ist=(imax-imin)/height;
791  for(x=xØ;x<width;++x)
792  {
793     for(re=rmin+x*rst,y=yØ;y<height-1;++y)
794     {
795        im=imin+y*ist;
796        n=yØ=Ø;   /* nach dem Laden yØ zuruecksetzen */
797        /* wurde Menu angewaehlt ? */
798        if((WorkMessage=GetMsg(WorkWindow->UserPort))!=Ø)
799        {
8ØØ           ReplyMsg(WorkMessage);
8Ø1           /* Frame bei 3D Modus nicht moeglich!! */
8Ø2           if((WorkMessage->Class!=MOUSEBUTTONS)(dreidf==Ø))
8Ø3              i=whats_up(WorkMessage->Class,WorkMessage->Code);
8Ø4           else i=Ø;
8Ø5           if(i==2) return();
8Ø6           if(i==3)
8Ø7           {
8Ø8              if(frame(MyScreen,rmin,imin,rmax,imax)==Ø)
8Ø9              {
81Ø                 /* neue Koordinaten wurden eingegeben  */
811                 SetRast(rp,Ø);  /* Screen loeschen!    */
812                 rmin=rxmin,rmax=rxmax;
813                 imin=rymin,imax=rymax;
814                 rst=(rmax-rmin)/width,ist=(imax-imin)/height;
815                 y=height,x=Ø;   /* neues Bild starten  */
816                 break; break;
817              }
818           }
819        }
82Ø        if(dreidf==Ø)
821        {
822           SetAPen(rp,anzcol-1);
823           WritePixel(rp,x,y);
824        }
825        switch(funktion)
826        {
827           case Ø: /* Newton 3 */
828              newton3();
829              break;
83Ø           case 1: /* Newton 4 */
831              newton4();
832              break;
833           case 2: /* Newton 5 */
834              newton5();
835              break;
836           case 3: /* Mandel */
837              mandel();
838              break;
839           case 4: /* z^3-c */
84Ø              z_3();
841              break;
842           case 5: /* z^4-c */
843              z_4();
844              break;
845           case 6: /* z^5-c */
846              z_5();
847              break;
848           case 7: /* Julia-Menge */
849              julia();
85Ø              break;
851           case 8: /* 'Julia-Menge' z^3-c */
852              julia_3();
853              break;
```

```
854           case 9: /* 'Julia-Menge' z^4-c */
855              julia_4();
856              break;
857           case 10: /* 'Julia-Menge' z^5-c */
858              julia_5();
859              break;
860           case 11: /* Erweiterung */
861              erweiterung();
862              break;
863        }
864        col=n%(anzcol-2)+1;
865        if((n==nmaxn<noff)&&dreidf==0) col=0; /* nmax -> schwarz */
866        if(dreidf==2) n=nmax-n; /* Mode invertiert */
867        n*=fak;
868        SetAPen(rp,col);
869        if(dreidf==0)
870           WritePixel(rp,x,y);
871        else
872        { /* 3D Ausgabe */
873           xs=x+fxoff+(height-y);
874           ys=fyoff+y;
875           ye=ys-n-yoff;
876           if(ye<0) ye=0;
877           Move(rp,xs,ys);
878           Draw(rp,xs,ye);
879           SetAPen(rp,anzcol-col);
880           WritePixel(rp,xs,ye);
881        }
882     }
883  }
884  /* Bild fertig berechnet. Warten bis Menue angewaehlt */
885  while((i=ende())==0);
886  if(i==2) fractal();
887 }
888
889
890 /***********************/
891 /* newton5             */
892 /* z^5-1=0             */
893 /***********************/
894
```

```
895 newton5()
896 {
897  for(n=0,re1=re,im1=im;n<nmax;++n)
898  {
899     if((re1+im1)!=0)
900     {
901        h1=re1*re1-im1*im1;
902        h2=h1*h1-4*re1*re1*im1*im1;
903        h1=4*h1*re1*im1;
904        h3=h1*h1+h2*h2;
905        if(h3<MIN)          /* Absturzgefahr */
906           break;
907        re2=(4*re1+h2/h3)/5,im2=(4*im1-h1/h3)/5;
908        re1=re2,im1=im2;
909     }
910     else /* Null? */
911        break;
912     /* Abstaende des neuen Folgenglieds zu den einzelnen */
913     /* Attraktoren, das heisst Nullstellen.              */
914     for(i=0;i<5;++i)
915     {
916        h1=re1-nsre[i],h2=im1-nsim[i];
917        d[i]=h1*h1+h2*h2;
918     }
919     /* Ist der Attraktor erreicht? */
920     if(d[0]<emind[1]<emind[2]<emind[3]<emind[4]<emin)
921        break;
922     /* Attraktor Unendlich? */
923     if(d[0]>emaxd[1]>emaxd[2]>emaxd[3]>emaxd[4]>emax)
924        break;
925  }
926 }
927
928
929 /***********************/
930 /* newton4             */
931 /* z^4-1=0             */
932 /***********************/
933
934 newton4()
935 {
```

```
936  for(n=Ø,re1=re,im1=im;n<nmax;++n)
937  {
938     h1=re1*re1-im1*im1,h2=h1*re1-2*re1*im1*im1;
939     h1=h1*im1+2*re1*re1*im1;
94Ø     h3=h1*h1+h2*h2;
941     if(h3>MIN)
942     {
943        re2=(3*re1+h2/h3)/4,im2=(3*im1-h1/h3)/4;
944        re1=re2,im1=im2;
945     }
946     else /* Null? */
947        break;
948     /* Abstaende des neuen Folgenglieds zu den einzelnen */
949     /* Attraktoren, das heisst Nullstellen.            */
95Ø     for(i=Ø;i<4;++i)
951     {
952        h1=re1-nsre[i],h2=im1-nsim[i];
953        d[i]=h1*h1+h2*h2;
954     }
955     /* Ist der Attraktor erreicht? */
956     if(d[Ø]<emind[1]<emind[2]<emind[3]<emin)
957        break;
958     /* Attraktor Unendlich?        */
959        if(d[Ø]>emaxd[1]>emaxd[2]>emaxd[3]>emax)
96Ø        break;
961  }
962 }
963
964
965 /***********************/
966 /* newton3             */
967 /* z^3-1=Ø             */
968 /***********************/
969
97Ø newton3()
971 {
972  for(n=Ø,re1=re,im1=im;n<nmax;++n)
973  {
974     h1=re1*re1+im1*im1;
975     if(h1>MIN)
976     {
977        re2=(2*re1+(re1*re1-im1*im1)/(h1*h1))/3;
978        im2=(2*im1-(2*re1*im1)/(h1*h1))/3;
979        re1=re2,im1=im2;
98Ø     }
981     else /* Null? */
982        break;
983     /* Abstaende des neuen Folgenglieds zu den einzelnen */
984     /* Attraktoren, das heisst Nullstellen.            */
985     for(i=Ø;i<3;++i)
986     {
987        h1=re1-nsre[i],h2=im1-nsim[i];
988        d[i]=h1*h1+h2*h2;
989     }
99Ø     /* Ist der Attraktor erreicht?  */
991     if(d[Ø]<emind[1]<emind[2]<emin)
992        break;
993     /* Attraktor Unendlich?         */
994     if(d[Ø]>emaxd[1]>emaxd[2]>emax)
995        break;
996  }
997 }
998
999
1ØØØ /***********************/
1ØØ1 /* mandel              */
1ØØ2 /* z^ 2-c=z            */
1ØØ3 /***********************/
1ØØ4
1ØØ5 mandel()
1ØØ6 {
1ØØ7  for(n=re1=im1=Ø,cre=re,cim=im;n<nmax;++n)
1ØØ8  {
1ØØ9     im2=2*re1*im1-cim,re2=re1*re1-im1*im1-cre;
1Ø1Ø     im1=im2,re1=re2;
1Ø11     d[Ø]=im1*im1+re1*re1;
1Ø12     /* Ist der Attraktor erreicht? */
1Ø13     if(d[Ø]<emin)  break;
1Ø14     /* Attraktor Unendlich?        */
1Ø15     if(d[Ø]>emax)  break;
1Ø16  }
1Ø17 }
1Ø18
```

```
1Ø19
1Ø2Ø /************************/
1Ø21 /* z^ 3-c=z             */
1Ø22 /************************/
1Ø23
1Ø24 z_3()
1Ø25 {
1Ø26  for(n=re1=im1=Ø,cre=re,cim=im;n<nmax;++n)
1Ø27  {
1Ø28     im2=3*re1*re1*im1-im1*im1*im1-cim,re2=re1*re1*re1-3*im1*im1*re1-cre;
1Ø29     im1=im2,re1=re2;
1Ø3Ø     d[Ø]=im1*im1+re1*re1;
1Ø31     /* Ist der Attraktor erreicht?  */
1Ø32     if(d[Ø]<emin)  break;
1Ø33     /* Attraktor Unendlich?         */
1Ø34     if(d[Ø]>emax)  break;
1Ø35  }
1Ø36 }
1Ø37
1Ø38
1Ø39 /************************/
1Ø4Ø /* z^ 4-c=z             */
1Ø41 /************************/
1Ø42
1Ø43 z_4()
1Ø44 {
1Ø45  for(n=re1=im1=Ø,cre=re,cim=im;n<nmax;++n)
1Ø46  {
1Ø47     h1=3*re1*re1*im1-im1*im1*im1,h2=re1*re1*re1-3*im1*im1*re1;
1Ø48     im2=re1*h1+im1*h2-cim,re2=re1*h2-im1*h1-cre;
1Ø49     im1=im2,re1=re2;
1Ø5Ø     d[Ø]=im1*im1+re1*re1;
1Ø51     /* Ist der Attraktor erreicht?  */
1Ø52     if(d[Ø]<emin)  break;
1Ø53     /* Attraktor Unendlich?         */
1Ø54     if(d[Ø]>emax)  break;
1Ø55  }
1Ø56 }
1Ø57
1Ø58
```

```
1Ø59 /************************/
1Ø6Ø /* z^ 5-c=z             */
1Ø61 /************************/
1Ø62
1Ø63 z_5()
1Ø64 {
1Ø65  for(n=re1=im1=Ø,cre=re,cim=im;n<nmax;++n)
1Ø66  {
1Ø67     h1=3*re1*re1*im1-im1*im1*im1,h2=re1*re1*re1-3*im1*im1*re1;
1Ø68     h3=re1*h1+im1*h2,h4=re1*h2-im1*h1;
1Ø69     im2=re1*h3+im1*h4-cim,re2=re1*h4-im1*h3-cre;
1Ø7Ø     im1=im2,re1=re2;
1Ø71     d[Ø]=im1*im1+re1*re1;
1Ø72     /* Ist der Attraktor erreicht?  */
1Ø73     if(d[Ø]<emin)  break;
1Ø74     /* Attraktor Unendlich?         */
1Ø75     if(d[Ø]>emax)  break;
1Ø76  }
1Ø77 }
1Ø78
1Ø79
1Ø8Ø /************************/
1Ø81 /* Julia-Mengen         */
1Ø82 /* z^ 2-c=z             */
1Ø83 /************************/
1Ø84
1Ø85 julia()
1Ø86 {
1Ø87  /* cre und cim wie vorgewaehlt */
1Ø88  for(n=Ø,re1=re,im1=im;n<nmax;++n)
1Ø89  {
1Ø9Ø     im2=2*re1*im1-cim,re2=re1*re1-im1*im1-cre;
1Ø91     im1=im2,re1=re2;
1Ø92     d[Ø]=im1*im1+re1*re1;
1Ø93     /* Ist der Attraktor erreicht?  */
1Ø94     if(d[Ø]<emin)  break;
1Ø95     /* Attraktor Unendlich?         */
1Ø96     if(d[Ø]>emax)  break;
1Ø97  }
1Ø98 }
1Ø99
```

```
/***********************/
/* 'Julia'-Mengen      */
/* z^3-c=z             */
/***********************/

julia_3()
{
 /* cre und cim wie vorgewaehlt */
 for(n=0,re1=re,im1=im;n<nmax;++n)
 {
    im2=3*re1*re1*im1-im1*im1*im1-cim,re2=re1*re1*re1-3*im1*im1*re1-cre;
    im1=im2,re1=re2;
    d[0]=im1*im1+re1*re1;
    /* Ist der Attraktor erreicht?  */
    if(d[0]<emin)  break;
    /* Attraktor Unendlich?         */
    if(d[0]>emax)  break;
 }
}


/***********************/
/* 'Julia'-Mengen      */
/* z^4-c=z             */
/***********************/

julia_4()
{
 /* cre und cim wie vorgewaehlt */
 for(n=0,re1=re,im1=im;n<nmax;++n)
 {
    h1=3*re1*re1*im1-im1*im1*im1,h2=re1*re1*re1-3*im1*im1*re1;
    im2=re1*h1+im1*h2-cim,re2=re1*h2-im1*h1-cre;
    im1=im2,re1=re2;
    d[0]=im1*im1+re1*re1;
    /* Ist der Attraktor erreicht?  */
    if(d[0]<emin)  break;
    /* Attraktor Unendlich?         */
    if(d[0]>emax)  break;
 }
}


/***********************/
/* 'Julia'-Mengen      */
/* z^5-c=z             */
/***********************/

julia_5()
{
 /* cre und cim wie vorgewaehlt */
 for(n=0,re1=re,im1=im;n<nmax;++n)
 {
    h1=3*re1*re1*im1-im1*im1*im1,h2=re1*re1*re1-3*im1*im1*re1;
    h3=re1*h1+im1*h2,h4=re1*h2-im1*h1;
    im2=re1*h3+im1*h4-cim,re2=re1*h4-im1*h3-cre;
    im1=im2,re1=re2;
    d[0]=im1*im1+re1*re1;
    /* Ist der Attraktor erreicht?  */
    if(d[0]<emin)  break;
    /* Attraktor Unendlich?         */
    if(d[0]>emax)  break;
 }
}


/***********************/
/* Erweiterung         */
/***********************/

erweiterung()
{
 /* cre und cim wie vorgewaehlt */
 for(n=0,re1=re,im1=im;n<nmax;++n)
 {
    d[0]=im1*im1+re1*re1;
    /* Ist der Attraktor erreicht?  */
    if(d[0]<emin)  break;
    /* Attraktor Unendlich?         */
    if(d[0]>emax)  break;
 }
}
```

```
/* Warten und Testen ob Menu richtig angewaehlt! */
/* Rueckkehr mit Ø -> Cancel nicht angewaehlt   */
/*             1 -> Cancel angewaehlt          */
/*             2 -> Bereich angewaehlt         */
ende()
{
 while((WorkMessage=GetMsg(WorkWindow->UserPort))==Ø);
 ReplyMsg(WorkMessage);
 if((WorkMessage->Class!=MOUSEBUTTONS)(dreidf==Ø))
    i=whats_up(WorkMessage->Class,WorkMessage->Code);
 else i=Ø;
 if(i==2) return(1);
 if(i==3)
 {
    if(frame(MyScreen,rmin,imin,rmax,imax)==Ø)
    {
       /* neue Koordinaten wurden eingegeben  */
       /* SetRast(rp,Ø);  /* Screen loeschen! */
       rmin=rxmin,rmax=rxmax;
       imin=rymin,imax=rymax;
       rst=(rmax-rmin)/width,ist=(imax-imin)/height;
       x=y=xØ=yØ=Ø;
       return(2);
    }
 }
 return(Ø);
}
```

```
/*********************************************************************/
/* Frame-Routine zur Auswahl eines Bereichs mit der Maus             */
/*                                                                   */
/* Uebergabe von:                                                    */
/* scr -> Pointer auf den Screen                                     */
/* xmin,ymin,xmax,ymax -> relative Groesse des Screens               */
/*                                                                   */
/* Fehler:                                                           */
/* Ø -> kein Fehler                                                  */
/* 1 -> kein Screen-Pointer wurde uebergeben                         */
/* 2 -> frame() wurde mit unsinnigen Maus-Koordinaten aufge-         */
/*      rufen!                                                       */
/* 3 -> geframter Bereich zu klein!!                                 */
/*********************************************************************/
```

```

#include <exec/types.h>
#include <graphics/gfx.h>
#include <math.h>
#include <graphics/rastport.h>
#include <intuition/intuition.h>


#define MINX 5    /* Mindestgroesse in x-Richtung */
#define MINY 5    /* Mindestgroesse in y-Richtung */

/* extern float rxmin,rxmax,rymin,rymax;                        */
/* Variable fuer die Uebergabe muessen im Hauptprogramm  defi- */
/* niert sein!                                                  */


float rxmin,rxmax,rymin,rymax;

frame(scr,xmin,ymin,xmax,ymax)
struct Screen *scr;        /* Zeiger auf die Screen-Struktur   */
float xmin,ymin,ymax,xmax; /* relative Koordinaten des Screens */
{
 struct RastPort *rp;
 short scrWidth,scrHeight,xØ,yØ,x,y;
 unsigned char *port;

 port=12574721;            /* Port A des parallelen Ports      */
 if(scr==Ø)
    return(1);             /* Fehler: keine Pointer-Uebergabe  */
 rp=&(scr->RastPort);
 scrWidth=scr->Width;
 scrHeight=scr->Height;
 xØ=x=scr->MouseX;
 yØ=y=scr->MouseY;
 if((xØ<Ø)(xØ>scrWidth)(yØ>scrHeight))
    return(2);             /* frame() nicht auf dem Screen!!   */
 SetDrMd(rp,COMPLEMENT);   /* invertiert zeichnen!             */
 SetAPen(rp,Ø);
 while((*port&64)!=64)     /* solange der Button gedrueckt ist */
 {
    Rect(rp,xØ,yØ,x,y);    /* Viereck zeichnen                 */
    while((x==scr->MouseX)&&(y==scr->MouseY)&&((*port&64)!=64));
```

```
57      Rect(rp,xØ,yØ,x,y);     /* Viereck loeschen                */
58      x=scr->MouseX;          /* neue Koordinaten                */
59      y=scr->MouseY;
6Ø      if((x<Ø)(x>scrWidth)(y>scrHeight)(y<Ø))
61      {
62         SetDrMd(rp,Ø);       /* wieder normal zeichnen          */
63         return(2);           /* frame() nicht auf dem Screen!!  */
64      }
65   }
66   SetDrMd(rp,Ø);             /* wieder normal zeichnen          */
67   /* Bereich zu klein? */
68   if(((x-xØ)<MINX)((y-yØ)<MINY))
69      return(3);
7Ø   /* Koordinaten des Frames berechnen */
71   rxmin=xmin+(float)xØ*(xmax-xmin)/(float)(scrWidth);
72   rxmax=xmin+(float)x*(xmax-xmin)/(float)(scrWidth);
73   rymin=ymin+(float)yØ*(ymax-ymin)/(float)(scrHeight);
74   rymax=ymin+(float)y*(ymax-ymin)/(float)(scrHeight);
75   return(Ø);
76 }
77
78
79 Rect(rp,xØ,yØ,x,y)           /* Rechteck zeichnen               */
8Ø struct RastPort *rp;
81 short x,y;
82 {
83    Move(rp,xØ,yØ);
84    Draw(rp,x,yØ);
85    Draw(rp,x,y);
86    Draw(rp,xØ,y);
87    Draw(rp,xØ,yØ);
88 }
89
```

Die Bilddaten werden beim Abspeichern eines Bildes als »*.info*«-File mit dem eigentlichen Bildfile abgespeichert. Diese Datei hat folgenden Aufbau:

```
struct INFOC          /* Info-Datei                          */
{
 float rmin;          /* xmin,xmax,ymin,ymax                 */
 float rmax;          /* rel. Koordinaten des Bildes         */
 float imin;
 float imax;
 float cre;           /* cre, cim fuer Julia-Mengen          */
 float cim;
 short mx;            /* mx,my augenblickliche Koordinaten   */
 short my;
 int   nmax;          /* Bild-Parameter                      */
 int   noff;
 float emin,emax;
 char  mode;          /* Berechnungsmodus?                   */
 char  res;           /* Aufloesung des Bildes               */
 char  dim;           /* 3D Bild?                            */
 char  half_brite;    /* HALF_BRITE Modus?                   */
 char  text[256];
}
```

Um die Koordinaten und andere beschreibende Daten diesem »*.info*«-File zu entnehmen ohne das Programm *fraktal.c* zu starten, hier zum Abschluß ein Programm, das diese Datei auswertet.

Programm: *showinfo.c*

```
 1 /* Ausgabe der Bild-Parameter */
 2
 3 #include <exec/types.h>
 4 #include <math.h>
 5 #include <libraries/dos.h>
 6
 7
 8 /* Definition des Info-Chunks */
 9 struct INFOC          /* Info-Datei                          */
1Ø {
11  float rmin;          /* xmin,xmax,ymin,ymax                 */
12  float rmax;          /* rel. Koordinaten des Bildes         */
13  float imin;
14  float imax;
15  float cre;           /* cre, cim fuer Julia-Mengen          */
```

```
16  float cim;
17  short mx;           /* mx,my augenblickliche Koordinaten  */
18  short my;
19  int   nmax;         /* Bild-Parameter                     */
20  int   noff;
21  float emin,emax;
22  char  mode;         /* Berechnungsmodus?                  */
23  char  res;          /* Aufloesung des Bildes              */
24  char  dim;          /* 3D Bild?                           */
25  char  half_brite;   /* HALF_BRITE Modus?                  */
26  char  text[256];
27 } MyInfo;
28
29 char name[100];
30 long file;
31
32
33 main(argc,argv)
34 int argc;
35 char *argv[];
36 {
37  if(argc<2)
38  {
39    printf("Eingabe: %s [Bild-File]\n",argv[0]);
40    exit(FALSE);
41  }
42  strcpy(name,argv[1]);
43  strcpy(name+strlen(name),".info");
44  if((file=Open(name,MODE_OLDFILE))==0)
45  {
46     printf("kein Info-File vorhanden\n");
47     exit(FALSE);
48  }
49  Read(file,&MyInfo,sizeof(MyInfo));
50  Close(file);
51  printf("File: %s\n\n",name);
52  printf("Koordinaten: ");
53  printf("[%f,%f][%f,%f]\n",MyInfo.rmin,MyInfo.rmax,MyInfo.imin,MyInfo.
54  printf("cre: %f  cim:%f\n",MyInfo.cre,MyInfo.cim);              imax);
55  printf("zuletzt berechneter Punkt: ");
56  printf("[%d,%d]\n",MyInfo.mx,MyInfo.my);
57  printf("nmax: %d  noff: %f\n",MyInfo.nmax,MyInfo.noff);
58  printf("emin: %f  emax: %f\n",MyInfo.emin,MyInfo.emax);
59  printf("Aufloesung: ");
60  switch(MyInfo.res) /* Aufloesung */
61  {
62     case 0: /* 320x250x5(6) */
63             printf("320x250\n");
64             break;
65     case 1: /* 640x250x4 */
66             printf("640x250\n");
67             break;
68     case 2: /* 320x500x5(6) */
69             printf("320x500\n");
70             break;
71     case 3: /* 640x500x4 */
72             printf("640x500\n");
73             break;
74  }
75  printf("Half-Brite-Modus (64 Farben) ");
76  if(MyInfo.half_brite==0) printf("nicht ");
77  printf("gewaehlt.\n3-D Modus ");
78  if(MyInfo.dim==0) printf("nicht ");
79  printf("gewaehlt.\nFunktion: ");
80  switch(MyInfo.mode) /* Funktion */
81  {
82     case 0: /* z^3-1 */
83             printf("z^3-1 (Newton-Verfahren)\n");
84             break;
85     case 1: /* z^4-1 */
86             printf("z^4-1 (Newton-Verfahren)\n");
87             break;
88     case 2: /* z^5-1 */
89             printf("z^5-1 (Newton-Verfahren)\n");
90             break;
91     case 3: /* z^2-c */
92             printf("z^2-c (Apfelmaennchen)\n");
93             break;
94     case 4: /* z^3-c */
95             printf("z^3-c\n");
96             break;
97     case 5: /* z^4-c */
98             printf("z^4-c\n");
99             break;
```

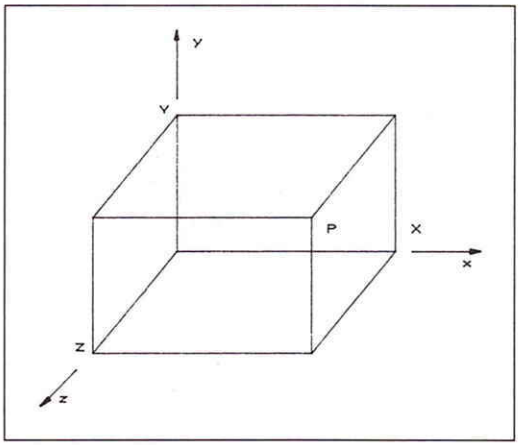
```
1ØØ     case 6: /* z^5-c */
1Ø1             printf("z^5-c\n");
1Ø2             break;
1Ø3     case 7: /* J z^2-c */
1Ø4             printf("z^2-c (Julia)\n");
1Ø5             break;
1Ø6     case 8: /* J z^3-c */
1Ø7             printf("z^3-c (J)\n");
1Ø8             break;
1Ø9     case 9: /* J z^4-c */
11Ø             printf("z^4-c (J)\n");
111             break;
112     case 1Ø: /* J z^5-c */
113             printf("z^5-c (J)\n");
114             break;
115     case 11: /* Erweiterung */
116             printf("Erweiterung\n");
117             break;
118   }
119   printf("\n%s\n",MyInfo.text);
12Ø   exit(TRUE);
121  }
```

Ein Ausdruck dieses Programms hat dann folgendes Aussehen:

```
File: Julia.pic1.info

Koordinaten: [-Ø.6Ø625Ø,-Ø.28125Ø][-Ø.3744ØØ,Ø.Ø192ØØ]
cre: -Ø.28ØØØØ  cim:-Ø.ØØ6ØØØ
zuletzt berechneter Punkt: [32Ø,249]
nmax: 2ØØ  noff: Ø.ØØØØØØ
emin: Ø.ØØØØØ1  emax: 1ØØØØØØ.ØØØØØØ
Aufloesung: 32Øx25Ø
Half-Brite-Modus (64 Farben) gewaehlt.
3-D-Modus nicht gewaehlt.
Funktion: z2-c (Julia)

Komplexe Dynamische Systeme
```

7 Grundlagen dreidimensionaler Grafik

Auf dem Bildschirm eines Rechners ist es aus verständlichen Gründen nur möglich, zweidimensionale Bilder darzustellen. Wollen wir mit räumlichen Objekten arbeiten, müssen wir diese erst mathematisch definieren. Dazu benötigen wir eine sinnvolle Datenstruktur, in der wir die Koordinaten der Eckpunkte, deren Verbindungsvorschriften sowie andere wichtige Informationen für das dreidimensionale Objekt ablegen können. Dann brauchen wir Transformationsalgorithmen, um die Objekte im Raum zu drehen und ihre Größe und Perspektive zu verändern. Schließlich müssen wir die fertig arrangierte Szene (d.h. alle in einem Bild auftretenden Objekte) auf dem Bildschirm möglichst so abbilden, daß sie trotz zweidimensionaler Darstellung räumlich wirkt.



7-1 Punktdarstellung in karthesischen Koordinaten P(x, y, z)

7.1 Dreidimensionale Transformationen

Als erstes müssen wir verstehen, wie wir die notwendigen dreidimensionalen Transformationen ausführen. Dazu stellen wir uns zuerst das relativ einfache Problem der Drehung um den Winkel α eines Punktes `P(x,y,z)` um die `z`-Achse. Nach der Drehung erhalten wir dann einen Bildpunkt `P'(x',y',z')`.

Den Punkt `P` kann man außer in den karthesischen Koordinaten `x`, `y` und `z` (siehe Abbildung 7-1) auch in Polarkoordinaten bezüglich der x-y-Ebene charakterisieren. Dazu wird der Punkt `P` auf die x-y-Ebene projiziert. Der erhaltene Punkt wird durch seinen Abstand `d` zum Ursprung und dem Winkel χ beschrieben. Der Winkel χ ist zwischen der x-Achse und der Verbindungsgeraden vom Ursprung zum projizierten Punkt zu finden (siehe Abbildung 7-2).

$$\chi = \arctan(\frac{y}{x})$$

$$d = x^2 + y^2$$

Drehen wir nun `P` um den Winkel α, so schließt der Bildpunkt `P'` mit der x-Achse einen Winkel von $\alpha+\chi$ ein und behält den Abstand `d` zum Ursprung (siehe Abbildung 7-3).

$$\chi' = \chi + \alpha$$

$$\Rightarrow \tan(\chi') = \tan(\chi+\alpha) = \frac{y'}{x'}$$

$$d = x'^2 + y'^2 = x^2 + y^2$$

Aus diesen beiden Gleichungen erhalten wir:

$$x' = x*\cos(\alpha) - y*\sin(\alpha)$$
$$y' = +x*\sin(\alpha) + y*\cos(\alpha)$$

Die z-Koordinate von P und P' ändert sich natürlich nicht, also

$$z' = z$$

Damit haben wir drei Formeln hergeleitet, die uns die Möglichkeit geben, einen Punkt um die z-Achse zu drehen. Analog erhalten wir die Drehung um die x- und y-Achse. Die Formeln für eine Verschiebung und eine Skalierung sind sogar noch einfacher herzuleiten. In der Mathematik gibt es nun eine Schreibweise für Gleichungssysteme, die Formeln allgemeiner und übersichtlicher darstellt. Dies gelingt mit einer sogenannten Matrix. Unsere drei Gleichungen von oben können wir auch wie folgt schreiben:

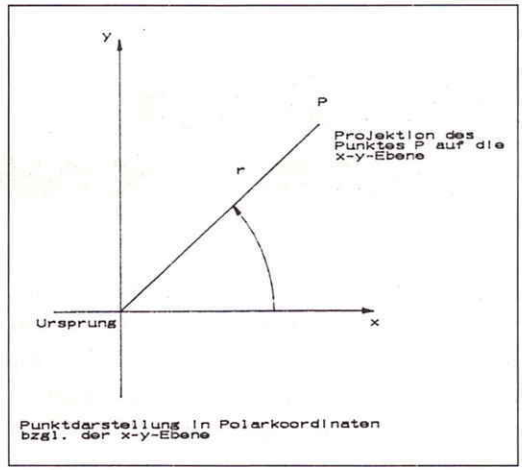
$$x' = x*\cos(\alpha) - y*\sin(\alpha) + 0$$
$$y' = +x*\sin(\alpha) + y*\cos(\alpha) + 0$$
$$z' = 0 + 0 + z$$

In der Matrix-Schreibweise sehen die Formeln etwas anders aus:

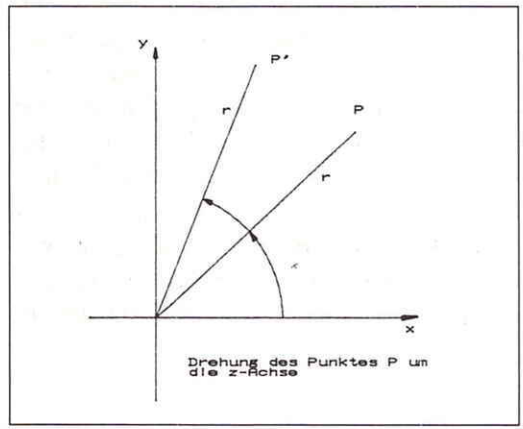
$$[x'\ y'\ z'] = [x\ y\ z] * \begin{pmatrix} \cos\alpha & \sin\alpha & 0 \\ +\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Matrix Rot_z

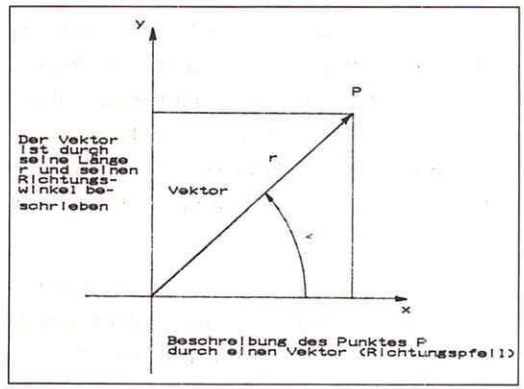
[x' y' z'] ist dabei der Bildvektor (Bildpunkt P') und [x y z] der Startvektor (Punkt P). Ein Vektor ist ein Richtungspfeil einer bestimmten Länge und Richtung. Deshalb können wir einen Punkt durch einen Vektor beschreiben (siehe Abbildung 7-4). Wir erkennen nun, wie ein Vektor mit einer Matrix multipliziert wird:



7-2



7-3



7-4

$$[x\ y\ z] * \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} = [x'\ y'\ z']$$

ergibt

$$x' = a_{11}*x + a_{12}*y + a_{13}*z$$
$$y' = a_{21}*x + a_{22}*y + a_{23}*z$$
$$z' = a_{31}*x + a_{32}*y + a_{33}*z$$

Wir können also ein Gleichungssystem wesentlich einfacher als Matrizenmultiplikation mit einem Vektor beschreiben. Dies ist natürlich nicht der einzige Grund. Mit dieser Schreibweise ist es vor allem möglich, Regeln zum Rechnen mit Gleichungssystemen einzuführen. Da dies kein Mathematiklehrbuch sein soll, sei hier zur Matrizenrechnung auf die einschlägigen Mathematikbücher verwiesen. Zum Schluß nun noch das Vorgehen bei der Multiplikation von zwei Matrizen `A` und `B`:

$$A * B = C$$

$$\begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}$$

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \quad \begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix}$$

mit

$$c_{11} = a_{11}*b_{11} + a_{12}*b_{21}$$
$$c_{21} = a_{21}*b_{11} + a_{22}*b_{21}$$
$$c_{12} = a_{11}*b_{12} + a_{12}*b_{22}$$
$$c_{22} = a_{21}*b_{12} + a_{22}*b_{22}$$

Die Matrixmultiplikation `A*B` erfolgt also analog zur Multiplikation einer Matrix mit einem Vektor. Das Ergebnis `C` ist dann wieder eine Matrix.

Translation

$$[x'\ y'\ z'\ 1] = [x\ y\ z\ 1] * \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ T_x & T_y & T_z & 1 \end{pmatrix}$$

Die Ausführung dieser Matrizenoperation auf den Punkt `P(x,y,z)` bedeutet eine Verschiebung dieses Punktes um den Translationsvektor $T(T_x,T_y,T_z)$. Als Ergebnis erhalten wir den verschobenen Punkt `P'(x',y',z')`.

Drehung

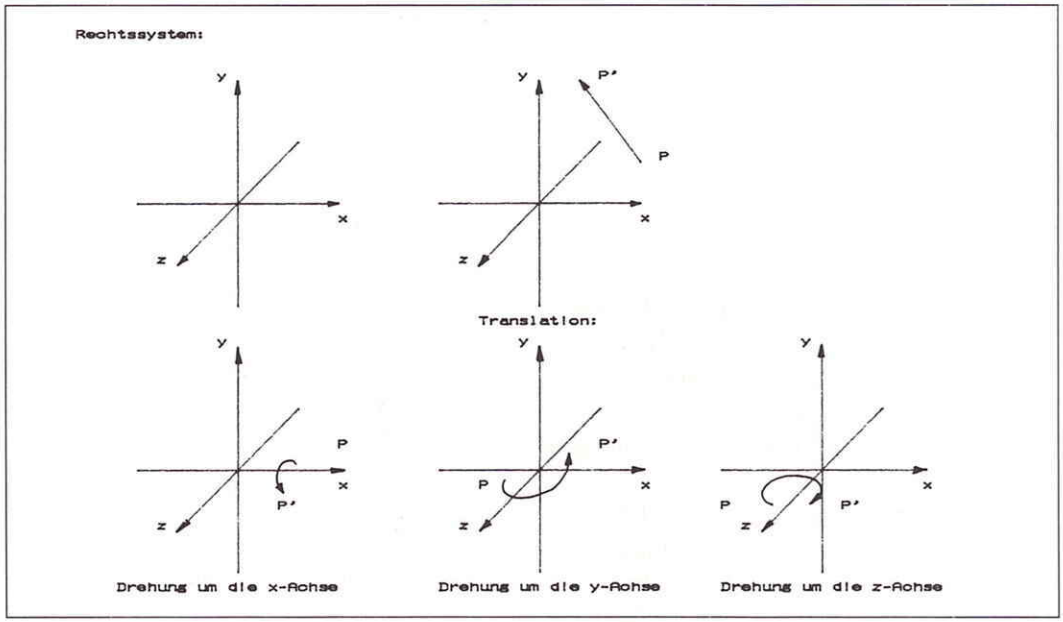
Bei der Drehung wird es schon etwas schwieriger als bei einer einfachen Transformation. Um einen Körper in eine beliebige andere Raumlage zu bewegen, reicht natürlich eine Verschiebung nicht aus, wir müssen ihn außerdem noch drehen. Dies können wir als unabhängige Drehungen um die x-, y- und z-Achse des Systems auffassen. Für jede dieser Teildrehungen kann dann wieder analog zur Translation eine Matrixoperation angegeben werden:

Drehung um die x-Achse:

$$[x'\ y'\ z'\ 1] = [x\ y\ z\ 1] * \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Drehung um die y-Achse:

$$[x'\ y'\ z'\ 1] = [x\ y\ z\ 1] * \begin{pmatrix} \cos\alpha & 0 & \sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\alpha & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



7-5

Drehung um die z-Achse:

$$[x'\ y'\ z'\ 1] = [x\ y\ z\ 1] * \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 & 0 \\ \sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Die Zerlegung einer Drehung um eine beliebige Achse im Raum ist deshalb sinnvoll, weil die drei dabei eventuell notwendigen Drehungen um x-, y- und z-Achse sehr viel einfacher sind. Für diejenigen, die schon mit Matrizen gearbeitet haben, eine Anmerkung: Zu den hier angeführten Matrizenoperationen existiert im allgemeinen die Inversion. Eine Multiplikation des Bildpunktes P' mit der inversen Matrix bedeutet, daß wir als P'' wieder P erhalten. Die Inversion einer Operation macht diese also wieder rückgängig. Im Falle einer Drehung um die z-Achse erhalten wir dann zum Beispiel die inverse Matrix:

$$[x'\ y'\ z'\ 1] = [x\ y\ z\ 1] * \begin{pmatrix} \cos\alpha & \sin\alpha & 0 & 0 \\ -\sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Dies folgt daraus, daß die Umkehrung einer Drehung um den Winkel α nichts anderes als ein Zurückdrehen um $-\alpha$ bedeutet. Mit $\sin(-\alpha)=-\sin\alpha$ und $\cos(-\alpha)=\cos\alpha$ erhalten wir dann aus der ursprünglichen Matrix ihre Inversion. Bei den Matrizen, die man bei geometrischen Anwendungen vorfindet, ist es also kein Problem die Inversion zu finden.

Skalierung

Außer den angesprochenen Transformationen ist es noch möglich, eine Skalierung vorzunehmen. Die dazu notwendige Operation lautet:

$$[x'\ y'\ z'\ 1] = [x\ y\ z\ 1] * \begin{pmatrix} S_x & 1 & \emptyset & \emptyset \\ \emptyset & S_y & 1 & \emptyset \\ \emptyset & \emptyset & S_z & \emptyset \\ \emptyset & \emptyset & \emptyset & 1 \end{pmatrix}$$

S_x, S_y und S_z geben die Skalierung in x-, y- und z-Richtung an. Mit einer Skalierung ist es zum Beispiel möglich, einen Körper in verschiedene Richtungen »schrumpfen« zu lassen oder »aufzublasen«.

Verknüpfung der elementaren Transformationen

Um nun einen Körper in eine beliebige neue Raumlage zu bringen, müssen wir, nachdem die Umwandlung in Teilschritte der obigen Art zerlegt wurde, die Matrizenoperationen mehrfach hintereinander ausführen. Wollen wir zum Beispiel zuerst um die x-Achse drehen (M_1), und dann um $T(T_x,T_y,T_z)$ verschieben (M_2), so geschieht dies folgendermaßen:

$$[x'\ y'\ z'\ 1] = [x\ y\ z] * M_1$$
$$[x''\ y''\ z''\ 1] = [x'\ y'\ z'\ 1] * M_2$$

oder unmittelbar

$$[x''\ y''\ z''\ 1] =$$
$$([x\ y\ z\ 1] * M_1) * M_2 = [x\ y\ z\ 1] * (M_1 * M_2)$$

Die Matrizenmultiplikation M_1*M_2 ergibt als Ergebnis wieder eine Matrix der gleichen Form. Wichtig ist, daß die Matrizenmultiplikation im allgemeinen nicht kommutativ ist, das heißt, M_1*M_2 ist nicht gleich M_2*M_1. In unserem Fall ist dies offentsichtlich, denn zuerst Drehen und dann Verschieben wird sicher zu einem anderen Ergebnis führen als Verschieben und dann Drehen. Wichtig ist also, daß wir die Reihenfolge der Operationen einhalten.

Sicher werden sich jetzt einige fragen, warum dieser Aufwand nötig ist. Er ist dadurch gerechtfertigt, daß wir alle räumlichen Transformationen allgemein einfach beschreiben können. In einem Programm (siehe 3D.c) brauchen wir nur noch für jede dieser Operationen ein Unterprogramm schreiben und behalten so die Übersicht.

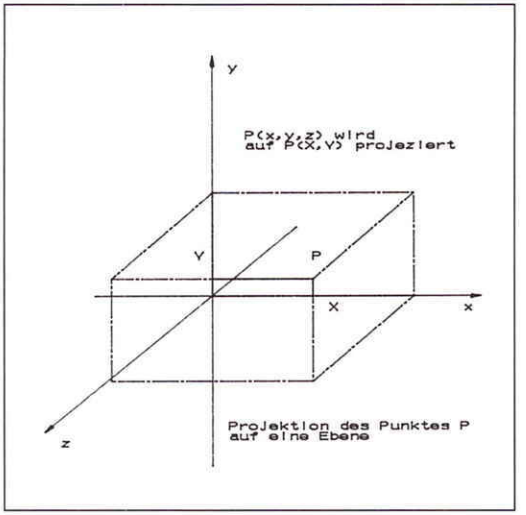
Für die Reihenfolge des Aufrufs der Operationen sind folgende Kriterien zu beachten: Wir unterscheiden ein Körpersystem und ein Weltkoordinatensystem. Haben wir zum Beispiel einen Würfel, so ist dessen Aussehen im Körperkoordinatensystem festgelegt. Wollen wir ihn nun im Weltkoordinatensystem (das was wir sehen!) plazieren, müssen wir die entsprechenden Verschiebungen und Drehungen ausführen.

- Eine Drehung im Körpersystem wird erreicht, wenn wir vor einer Verschiebung drehen.
- Eine Drehung um die Achsen des Weltkoordinatensystems erreichen wir, wenn zuerst verschoben und dann gedreht wird.

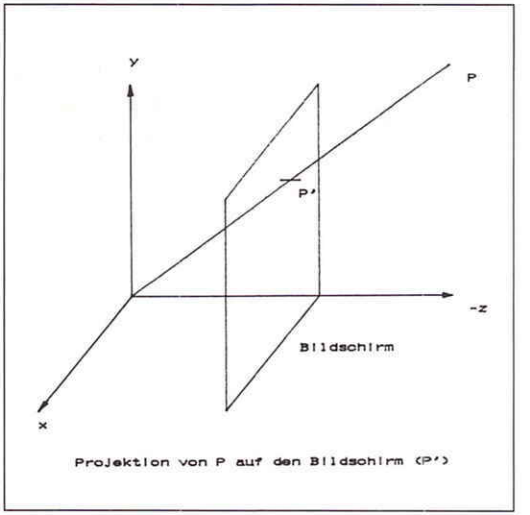
Wie gesagt, es kommt auf die Reihenfolge der Matrixmultiplikation an.

7.2 Projektion und Perspektive

Bisher haben wir uns damit beschäftigt, wie wir einen Raumpunkt in einen neuen überführen können. Wir wollen unser räumliches System aber nun auch auf dem Bildschirm



7-6



7-7-1

sichtbar machen – wir müssen also den Raum auf eine Ebene projizieren.

Die Projektion soll unter einem Winkel χ geschehen, das heißt, die z-Achse ist um den Winkel χ gegenüber der x-Achse geneigt. Dann erhalten wir die Bildkoordinaten in der Ebene des projizierten Punktes $P_p(X,Y)$:

```
X = x - z * cos(χ)
Y = y - z * sin(χ)
```

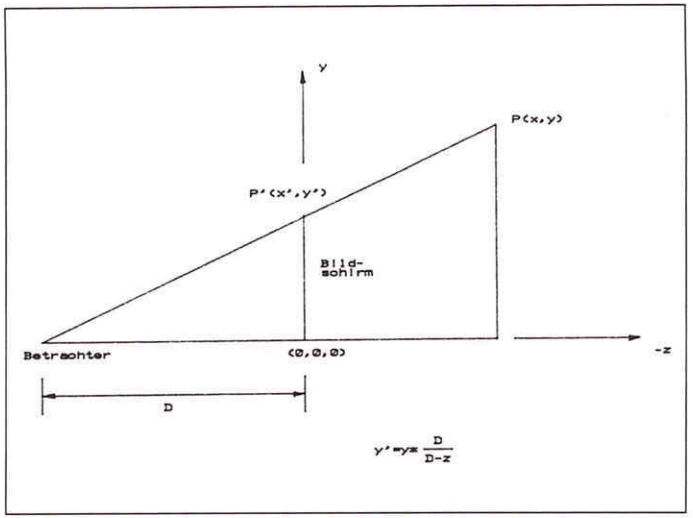
Diese Projektion wirkt aber ein wenig unwirklich – die Perspektive fehlt. Wir sind es gewohnt, weiter entfernte Objekte kleiner als nähere zu sehen. Ein Quader erhält erst dann eine Tiefe, wenn wir dies auch bei der Projektion auf die Ebene berücksichtigen.

Dazu betrachten wir nur die y-z-Ebene. Die Tiefe für unsere Projektion wird durch die z-Koordinate des zu projizierenden Punktes bestimmt. Wir stellen uns vor, daß der Betrachter sich in einem Abstand d vom Bildschirm auf der z-Achse befindet und in ihre negative Richtung schaut. Wir legen den Koordinatenursprung in die Bildschirmebene. Nach Abbildung 7–7 berechnet sich die y-Koordinate des projizierten Punktes P' aus dem folgenden Dreisatz:

$$\frac{y'}{y} = \frac{d}{(d-z)}$$

$$y' = y * \frac{d}{d-z}$$

Die Perspektive schlägt sich bei dieser Betrachtung mit einem Fluchtpunkt, der auf der z-Achse liegt, nur in der y-Koordinate nieder. Wir erkennen, daß die y-Koordinate eines hinter dem Bildschirm liegenden Punktes kleiner wird, wogegen dies bei einem Punkt vor der Bildebene genau umgekehrt ist. Kritisch wird es, wenn der Punkt näher beim Betrachter liegt, denn dann wird die y-Koordinate des projizierten Punktes immer größer. Die Berücksichtigung der Tiefe sollte nicht nur in y-Richtung erfolgen, sondern sich auch in der x-Koordinate des Bildpunktes auswirken (Sichtpyramide!). Wir verwen-



7-7-2

den die gleichen Formeln wie bei der y-Koordinate, nur daß y durch x zu ersetzen ist:

$$x' = x * \frac{d}{d-z}$$

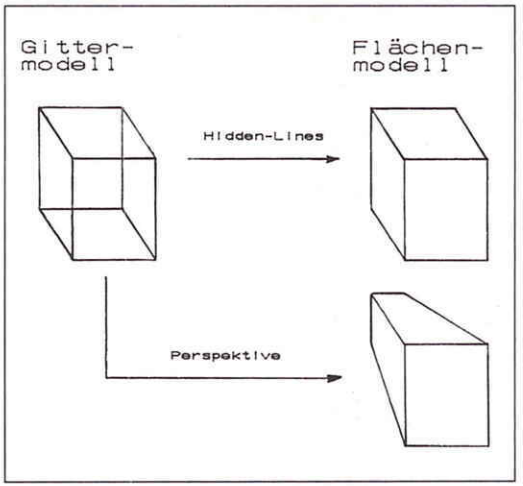
Ein häufig auftretender Programmierfehler (gerade auch bei den rekursiven Folgen, die wir noch kennenlernen werden) ist auf unerwünschte Rückkopplungen zurückzuführen. Zum Beispiel muß bei einer Drehung um eine Achse darauf geachtet werden, daß die neu berechnete x-Koordinate nicht zur Berechnung der y-Koordinate herangezogen wird. Machen wir dies dennoch, so treten Verzerrungen auf, die zu verstehen im fertigen Bild manchmal unmöglich scheint. Bei der Projektion dürfen wir in dieser Hinsicht nicht zuerst die Perspektive berechnen und dann die Punkte noch projizieren, da Verzerrungen sonst die Folge sind. Um ein vernünftiges Bild zu erhalten, dürfen wir also entweder nur projizieren oder nur mit Perspektivenberechnung arbeiten.

An dieser Stelle noch ein Hinweis: Beim Bewegen eines Punktes kann es passieren, daß seine Projektion nicht mehr auf dem Bildschirm darstellbar ist. Bei einem Punkt brauchen wir diesen einfach nicht zu zeichnen, aber zum Beispiel bei einer Geraden ist es möglich, daß ihre Endpunkte zwar außerhalb des Bildes liegen, sie aber dennoch teilweise sichtbar ist. In diesem Fall müssen die beiden gerade noch sichtbaren Punkte der Linie berechnet werden. Dann erst kann der sichtbare Teil gezeichnet werden. Bei Flächen stellt sich das Problem analog. All diese Aufgaben erfüllen sogenannte Clipping-Algorithmen, auf die ich aber hier nicht weiter eingehen will. In allen Programmen in denen sich ein solches Problem stellte, habe ich einfach, falls zum Beispiel die x-Koordinaten größer als die Breite *(Width)* des Screens war, diese gleich *Width* gesetzt. Damit ergeben sich zwar Verzerrungen, es ist aber sichergestellt, daß es zu keinem Absturz des Rechners kommen kann.

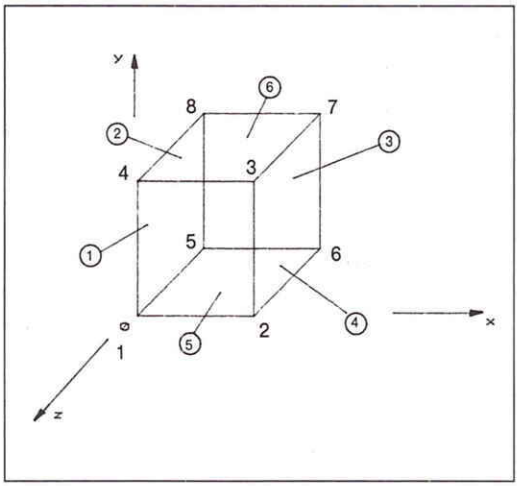
Die Bilder, die wir mit unseren bisherigen Erkenntnissen erhalten, zeichnen sich zwar nun schon durch Perspektive aus, aber da die normalerweise verdeckten Linien mitgezeichnet werden, sind die Bilder immer noch unwirklich. Wir müssen nun also versuchen die verdeckten Oberflächen und Linien zu unterdrücken!

7.3 Unterdrückung verdeckter Oberflächen und Linien

Jeder, der sich schon einmal mit dreidimensionaler Grafik beschäftigt hat, weiß, daß die Unterdrückung verdeckter Linien keine leichte Übung ist. Ein allgemeingültiger Algorithmus für alle Aufgaben ist nicht leicht zu erstellen. Wir wollen hier davon ausgehen, daß wir ein Flächenmodell eines Kör-



7-8



7-9

pers vorliegen haben, dessen Flächen sich nicht schneiden. (D.h., die Kanten schließen immer eine Fläche ein!) Um die verdeckten Flächen dieses Körpers zu unterdrücken, müssen wir feststellen, welche Flächen bzgl. des Betrachters weiter hinten liegen. Haben wir festgestellt, in welcher Reihenfolge die Flächen aufeinanderfolgen, zeichnen wir diese einfach nacheinander. Damit werden die hinten liegenden Flächen von den vorderen, später gezeichneten Flächen verdeckt. Wir erhalten ein räumliches Bild (Flächenmodell), wenn die Flächen außerden noch mit verschiedenen Farben gezeichnet werden. Wollen wir nur die sichtbaren Kanten sehen, so zeichnen wir einfach die Flächen mit Hintergrundfarbe und umranden sie. Dann erhalten wir ein Gitternetzmodell des Körpers (siehe Abbildung 7-8).

Aber wie erhalten wir nun eigentlich die Reihenfolge der Flächen? Dazu muß jeder Fläche ein Maß für ihre Tiefe zugeordnet werden. Vergleichen wir dann diese Werte, können wir feststellen, welche weiter hinten bzw. vorn liegt. Es sind mehrere Varianten denkbar, um das Maß der räumlichen Tiefe festzulegen. Wir wollen folgendermaßen vorgehen: Wir addieren die x-, y- und z-Koordinaten aller Eckpunkte der Fläche auf und dividieren durch ihre Anzahl. Wir erhalten als Ergebnis einen Punkt, der in der »Mitte« der Fläche liegt. Haben wir zum Beispiel ein Fünfeck, so erhalten wir

$$x' \frac{1}{5} = (x_1 + x_2 + x_3 + x_4 + x_5)$$

$$y' \frac{1}{5} = (y_1 + y_2 + y_3 + y_4 + y_5)$$

$$z' \frac{1}{5} = (z_1 + z_2 + z_3 + z_4 + z_5)$$

Wir wollen unsere Methode an einem Würfel überprüfen:

Der Würfel habe die acht Eckpunkte

```
1       2       3       4
[Ø,Ø,Ø] [1,Ø,Ø] [1,1,Ø] [Ø,1,Ø]
5       6       7       8
[Ø,Ø,-1] [1,Ø,-1] [1,1,-1] [Ø,1,-1]
```

Diese Punkte beschreiben bekanntlich sechs quadratische Flächen (siehe Abbildung 7-9).

```
Fläche 1: besteht aus Punkt  1,2,3,4
Fläche 2:                    1,5,8,4
Fläche 3:                    5,6,7,8
Fläche 4:                    6,2,3,7
Fläche 5:                    1,2,6,5
Fläche 6:                    4,3,7,8
```

(Die Punktreihenfolge muß beim Zeichnen eingehalten werden!)

Wir wissen aus unserer Vorstellung, daß die Flächen folgende Reihenfolge haben 5,2,3,1,4,6

Hier können die ersten und die letzten drei beliebig vertauscht werden! Nun wollen wir versuchen, nach unserem Verfahren das gleiche Ergebnis zu erhalten. Für jede Fläche berechnen wir den »Mittelpunkt«:

```
Fläche 1:    .5,.5,Ø
Fläche 2:     Ø,.5,-.5
Fläche 3:    .5,.5,-1
Fläche 4:     1,.5,-.5
Fläche 5:    .5,Ø,-.5
Fläche 6:    .5,1,-.5
```

Haben wir nun unser Koordinatensystem so angelegt, daß wir von oben auf den Körper schauen, also die oben vorgestellte Projektion verwenden, erhalten wir:

Test in z*-Richtung:*

Fläche 1 liegt vor Fläche 3, die Lage der restlichen Flächen aus dieser Untersuchung nicht zu ermitteln!

Test in x*-Richtung:*

Fläche 2 liegt hinter Fläche 4, die Lage von Fläche 5 und 6 kann immer noch nicht erkannt werden!

Test in y*-Richtung:*

Nun erkennen wir endlich auch die Reihenfolge von 5 und 6: Fläche 5 liegt unter 6!

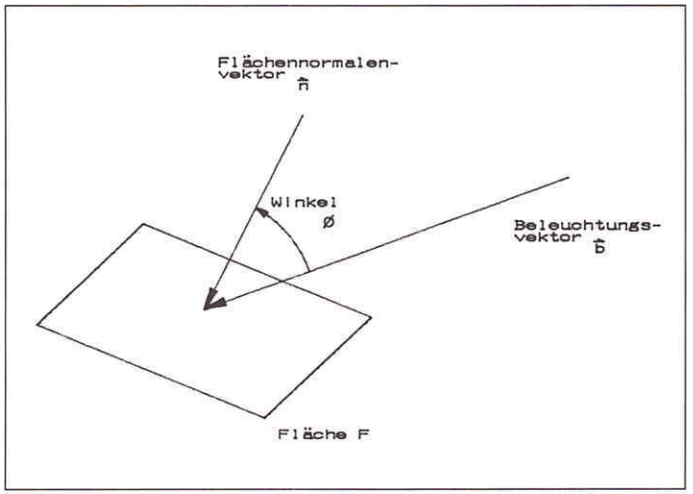
Da wir wissen, daß die Reihenfolge in der wir die hinten bzw. vorn liegenden Flächen zeichnen egal ist, erhalten wir damit das gleiche Ergebnis wie oben – unser Algorithmus funktioniert also. Wichtig ist, daß wir nicht vergessen, in allen drei Achsen die Koordinaten der Bewertungspunkte zu vergleichen. Aber wie gesagt, bei manchen Körpern kann dieses Vorgehen versagen, dann muß die Bewertung entsprechend erweitert werden. Der Algorithmus hat vor allem den Nachteil, daß er bei einer großen Anzahl von Flächen ziemlich langsam wird. Wenn es auf Schnelligkeit ankommt, wird man deshalb versuchen, einen einfachen »Pseudo-«Hidden-Lines Algorithmus zu finden.

Zeichnen wir ein Flächenmodell, so möchten wir doch eigentlich, daß die Flächen die Farbe bezüglich ihrer augenblicklichen Ausrichtung erhalten, das heißt, wir wollen die Farbe der Flächen in Abhängigkeit ihrer Lage zu einer »Lichtquelle« bestimmen.

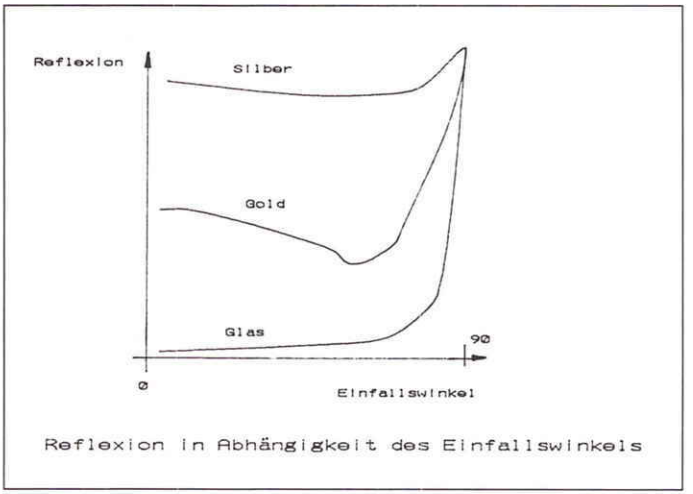
7.4 Schattierung

Wir gehen von folgender Annahme aus: Unsere Szene wird von parallelem Licht aus einer bestimmten Stelle im Raum beleuchtet. Wenn wir wissen, wie in diesem Fall die Farbe berechnet wird, ist es kein Problem mehr mit einer Lichtquelle zu arbeiten, die radiales Licht in alle Richtungen aussendet. (Der Beleuchtungsvektor wäre dann nicht mehr immer gleich, sondern würde von der Lage der untersuchten Fläche abhängen.)

Die Richtung des Lichts charakterisieren wir durch einen Beleuchtungsvektor



7-10



7-11

$b(b_x,b_y,b_z)$. Um die Lage der Fläche bezüglich $b(b_x,b_y,b_z)$ bewerten zu können, beschreiben wir sie ebenfalls durch einen Vektor. Dieser steht senkrecht auf der Fläche.

Die Länge sowohl des Beleuchtungs- als auch des Richtungsvektors ist für uns im Augenblick nicht von Bedeutung. Die Helligkeit der Fläche wird durch den Winkel χ, der durch diese beiden Vektoren eingeschlossen wird, festgelegt (siehe Abbildung 7-10). Wir müssen also diesen Winkel χ berechnen. Mit dem Skalarprodukt gelingt uns das ohne Probleme:

$$n * b = |n| * |b| * \cos(\chi)$$

also

$$\chi = \arccos\left(\frac{n * b}{|n| * |b|}\right)$$

(Vorsicht: Beim Arcus-Cosinus ist eventuell die Mehrdeutigkeit der Winkel zu berücksichtigen!) Da es für die Helligkeit keine Rolle spielt, ob die Fläche von links oder von rechts beleuchtet wird, verwenden wir nun im weiteren den Betrag des Winkels χ. Je kleiner $|\chi|$ ist, desto heller ist die Fläche, je größer $|\chi|$, desto dunkler erscheint sie. Angenommen wir haben n-Farben zur Verfügung, so erhalten wir:

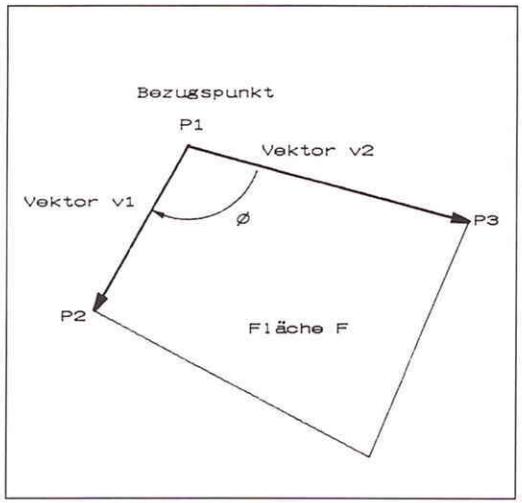
$$col = \left(1 - \frac{|\chi|}{90°}\right) * n$$

Diese Zuordnung erfolgt in diesem Fall linear, wir gehen also von einer Oberfläche aus, die unabhängig von der Beleuchtungsrichtung das gleiche Reflexionsverhalten zeigt. In der Wirklichkeit ist dies jedoch nicht der Fall. So ist es sogar möglich, verschiedene Materialien zu kennzeichnen (siehe Abbildung 7-11).

Gerade in dieser Hinsicht enttäuscht die Computerdarstellung verständlicherweise etwas, denn um eine wirklichkeitsgetreue Schattierung und damit Realitätsnähe zu erreichen, müßten wir unendlich viele Farbabstufungen zur Verfügung haben.

Um die obigen Berechnungen durchzuführen, ist ein wenig Vektorrechnung notwendig. Wir gehen von den Vektoren $b(b_x,b_y,b_z)$

7-12



und $n(n_x,n_y,n_z)$ aus. Das Skalarprodukt berechnet sich dann zu

$$b * n = b_x * n_x + b_y * n_y + b_z * n_z$$

und die Beträge zu

$$|b| = \sqrt{(b_x^2 + b_y^2 + b_z^2)}$$

$$|n| = \sqrt{(n_x^2 + n_y^2 + n_z^2)}$$

Nun können wir ohne Probleme den Winkel χ ausrechnen. Aber wie erhalten wir den Vektor n für eine bestimmte Fläche? Eine Fläche wird durch drei Punkte bestimmt (allgemein ist bei der Definition einer Fläche mit mehr als drei Punkten darauf zu achten, daß diese wirklich alle in einer Ebene liegen!). Wir verwenden also drei beliebige Eckpunkte der Fläche, im folgenden Punkt $P_1(x_1,y_1,z_1)$, $P_2(x_2,y_2,z_2)$ und $P_3(x_3,y_3,z_3)$. Nun berechnen wir die beiden Verbindungsvektoren v_1 und v_2 zwischen P_1,P_2 und P_1,P_3 (siehe Abbildung 7-12).

$$v_1 = \begin{pmatrix} x_2-x_1 \\ y_2-y_1 \\ z_2-z_1 \end{pmatrix} \qquad v_2 = \begin{pmatrix} x_3-x_1 \\ y_3-y_1 \\ z_3-z_1 \end{pmatrix}$$

Den Vektor n erhalten wir aus v_1 und v_2 mit Hilfe des Kreuz-Vektorprodukts. Für dieses Produkt gilt $v_1 \times v_2 = n$, wobei das Ergebnis n senkrecht auf v_1 und v_2 steht. Für den Betrag von n gilt

$$|n| = |v_1| * |v_2| * \sin(\chi)$$

(χ ist der Winkel den v_1 und v_2 einschließen.)

Die Definition des Vektorprodukts wird eigentlich mit einer Determinante vorgenommen, es gibt aber ein recht einfaches Merkschema:

$$n = v_1 \times v_2 =$$

$$\begin{matrix} v_{1x} \\ v_{1y} \\ v_{1z} \\ v_{1x} \\ v_{1y} \end{matrix} \times \begin{matrix} v_{2x} \\ v_{2y} \\ v_{2z} \\ v_{2x} \\ v_{2y} \end{matrix} = \begin{pmatrix} v_{1y}*v_{2z} - v_{1z}*v_{2y} \\ v_{1x}*v_{2x} - v_{1x}*v_{2z} \\ v_{1x}*v_{2y} - v_{1y}*v_{2x} \end{pmatrix}$$

Nehmen wir einen Beleuchtungsvektor b(1,1,1) an, so erhalten wir zum Beispiel für die Fläche 1 unseres Würfels:

$$v_1 = \begin{pmatrix} 1-ø \\ ø-ø \\ ø-ø \end{pmatrix} = \begin{pmatrix} 1 \\ ø \\ ø \end{pmatrix} \quad v_2 = \begin{pmatrix} ø-ø \\ 1-ø \\ ø-ø \end{pmatrix} = \begin{pmatrix} ø \\ 1 \\ ø \end{pmatrix}$$

$$n = v_1 \times v_2 = \begin{matrix} 1 \\ ø \\ ø \\ 1 \\ ø \end{matrix} \times \begin{matrix} ø \\ 1 \\ ø \\ ø \\ 1 \end{matrix} = \begin{pmatrix} ø-ø \\ ø-ø \\ 1-ø \end{pmatrix} = \begin{pmatrix} ø \\ ø \\ 1 \end{pmatrix}$$

Dieser Normalenvektor entspricht unserer Erwartung, denn wenn die Fläche durch einen Vektor in x- und einen in y-Richtung beschrieben wird, muß der senkrechte Vektor in z-Richtung weisen. (Hier noch eine Bemerkung: Außer daß n senkrecht auf v_1 und v_2 steht, bilden die drei Vektoren ein Rechtssystem!)

$$n*b = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} * \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} = 1$$

$|n| = 1$
$|b| = \sqrt{3}$

also

$$\chi = \arccos\left(\frac{1}{\sqrt{3}}\right) = 54.73°$$

Dieser Winkel ist nun Grundlage für die Bestimmung der Flächenfarbe. Angenommen wir haben 32 Farben zur Verfügung, so erhält die Fläche die Farbe

$$col = int\left(32 * \left(1 - \frac{54.73°}{90°}\right)\right) = 12$$

Zum Schluß möchte ich noch kurz darauf eingehen, wie eigentlich die Daten einer dreidimensionalen Szene anzulegen sind, um möglichst variabel zu bleiben und gleichzeitig die Übersicht zu bewahren.

7.5 Anlage einer Datenstruktur

Ich möchte hier drei Möglichkeiten zum Aufbau einer 3-D-Grafik vorstellen. Die erste ist so allgemein aufgebaut, daß mit ihr problemlos auch komplizierte räumliche Szenen realisiert werden können. Bei der zweiten beschränken wir unsere Darstellungsmöglichkeiten, erhalten dadurch aber eine wesentlich einfachere Datenstruktur. Bei der dritten schließlich erreichen wir eine räumliche Wirkung der Grafik nur noch durch geschicktes Zeichnen.

1) Allgemeine Datenstruktur für ein 3-D-Grafiksystem

Ein Grafik-Programmsystem hat die Aufgabe, nach der Erzeugung einer geeigneten Datenstruktur für die Objektbeschreibung, diese zu manipulieren und dann auf dem Bildschirm darzustellen. Wir erwarten von einem derartigen System, daß es einen möglichst hohen Grad an Ordnung, Übersichtlichkeit und Erweiterbarkeit aufweist und leicht zu realisieren ist.

Wir müssen uns zuerst einmal überlegen, wie zum Beispiel ein Würfel am besten zu beschreiben ist. Er besteht aus acht Eckpunkten, deren Koordinaten die Lage und das Aussehen beschreiben. Sie werden also für uns die wichtigste Grundlage sein. Nun müssen wir dem Rechner aber noch mitteilen, wie diese acht Punkte verbunden werden sollen. Hier gibt es zwei Möglichkeiten. Zum ersten geben wir immer ein Punktepaar an, das zu einer Kante verbunden wird. Dadurch nehmen wir uns aber die Möglichkeit, ein Flächenmodell des Körpers zu erstellen. Deshalb wird es besser sein, wir zerlegen das Objekt in Flächen und beschreiben, aus welchen Punkten diese bestehen.

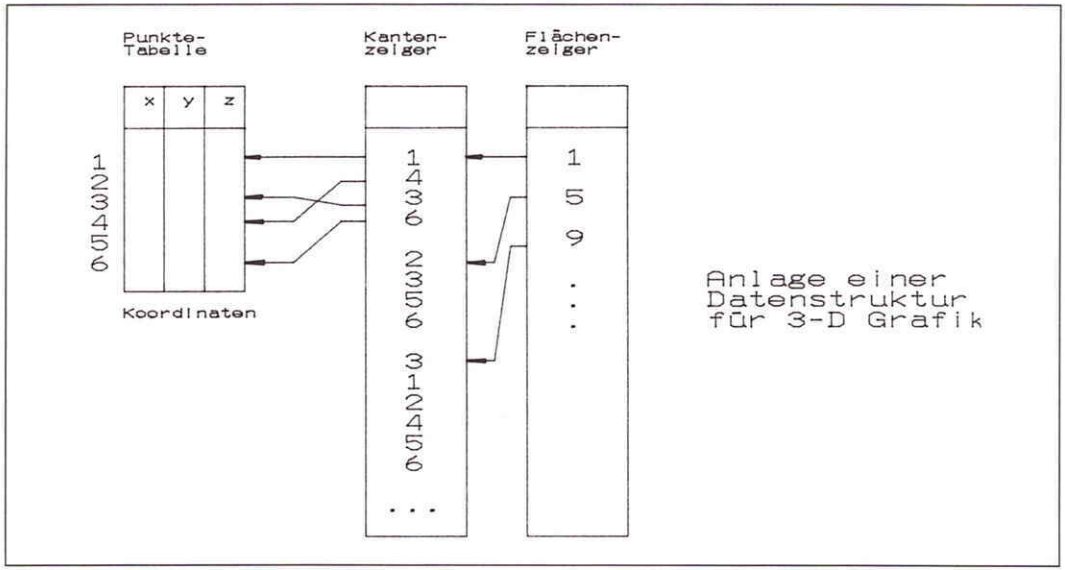
Als erstes brauchen wir also eine Punkte-Tabelle, in der die x-, y- und z-Koordinaten der Punkte abgelegt sind. Für unseren Würfel also zum Beispiel:

Punkte-Tabelle

	x	y	z	Koordinate
P_0	0	0	0	
P_1	1	0	0	
P_2	1	1	0	
P_3	0	1	0	
P_4	0	0	1	
P_5	1	0	1	
P_6	1	1	1	
P_7	0	1	1	

So, jetzt brauchen wir noch zwei weitere Tabellen, mit deren Hilfe wir festlegen, aus welchen Punkten eine Fläche besteht. In

7-13



der ersten, der sogenannten Kanten-Tabelle, stehen Zeiger auf die Punktkoordinaten der Punkte-Tabelle. Die zweite, die Flächen-Tabelle, enthält Zeiger auf das erste beschreibende Element der Kanten-Tabelle für diese Fläche. Das alles hört sich sehr kompliziert und undurchsichtig an, aber an unserem Beispiel wird das Vorgehen sicher klar (siehe auch Abbildung 7-13). Ich möchte den Aufbau der beiden Tabellen für den Würfel nur mit zwei Flächen ausführen, die restlichen Flächen werden dann analog definiert.

Wollen wir das Objekt zeichnen, müssen wir also der Reihe nach alle Flächen der Flächentabelle abarbeiten lassen. Zum Zeichnen der Fläche 0 (Zählung so gewählt, weil bei C das erste Feldelement 0 ist!) müssen wir also die Punkte verwenden, die in der Kantentabelle ab Position 0 bis 3 stehen. Dies sind die Punkte 0, 1, 2 und 3. Es ist verständlich, daß wir uns natürlich nicht nur auf Vierecke beschränken, sondern mit diesem System auch Vielecke problemlos definieren wollen. Für

Kanten-Tabelle			Flächen-Tabelle
Ø	Ø ($P_Ø$)	<- erste Fläche	Ø (Fläche Ø) Ø
1	1 (P_1)		1 (Fläche 1) 4
2	2 (P_2)		2 (Fläche 2) 8
3	3 (P_3)		3 ...
4	Ø ($P_Ø$)	<- zweite Fläche	
5	4 (P_4)		
6	7 (P_7)		
7	3 (P_3)		
8		<- dritte Fläche	
.			
.			
.			

Anwendungen, bei denen mit mehreren Körpern gearbeitet und diese getrennt von einander darstellbar sein sollen, ist es sinnvoll noch eine Körper-Tabelle anzulegen, in der angegeben wird, aus welchen Flächen ein Körper besteht. Sie hat den gleichen Aufbau wie die Flächentabelle und besteht aus Zeigern, die auf das erste beschreibende Element in der Flächentabelle weisen. Zum Schluß

noch eine Anmerkung: Das letzte Element der Flächen- und Körpertabelle, soweit vorhanden, muß auf das nächste freie Element der Kanten- bzw. Flächentabelle weisen, da wir aufgrund der aufeinanderfolgenden Zeiger ja erst wissen, bis zu welchem Element wir gehen müssen.

Wichtig: Die Koordinaten der Punkte-Tabelle haben nichts mit den Punktkoordinaten auf dem Bildschirm zu tun. Wir berechnen diese vor jeder Darstellung aus den Ursprungsdaten neu. Es werden also die Daten in der Punkte-Tabelle nicht verändert. Der Vorteil dabei ist, daß wir zum Beispiel auch nach mehrfachen Drehungen keine Verzerrungen aufgrund von Rechenungenauigkeit erhalten, sondern wieder in den Ausgangszustand zurückkehren können.

Als Anwendung dieser Datenstruktur habe ich das Programm *3-D.c* geschrieben, das deren Vielseitigkeit zeigen soll. Ein Nachteil liegt jedoch darin, daß die Anlage der Datenstruktur und ihre Verwaltung doch recht aufwendig ist. Auch der benötigte Speicherplatz ist beträchtlich, denn bei wachsender Flächenzahl steigt die Größe der Kanten-Tabelle sprunghaft an. Deshalb ist es durchaus sinnvoll, eine nicht so allgemeine, aber einfachere Struktur zu verwenden.

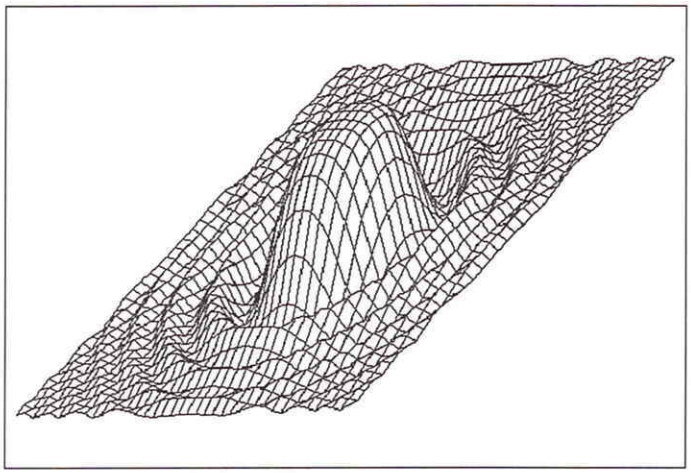
2) Datenstruktur mit Hilfe von Rasterung

Dazu legen wir in eine Ebene, zum Beispiel die `x-z`-Ebene des Koordinatensystems, eine Rasterung an. In der Praxis bedeutet das nichts anderes, als die Anlage eines zweidimensionalen Feldes `feld[n][n]`. In diesem Fall wäre unsere Ebene in `n*n=n`-Rasterpunkte aufgeteilt. Jedem dieser Rasterpunkte ordnen wir jetzt einen Wert zu, der die Höhe des Punktes bezüglich unserer gewählten Fläche angibt. Den Abstand der Rasterelemente geben wir uns durch Wahl der Größe der Grundfläche vor. Jetzt haben wir uns mit geringem Aufwand eine Datenstruktur angelegt. Diese Methode eignet sich besonders, um Landschaften und Flächen aufzubauen (siehe Abbildung 7-15). Sie hat den Vorteil, daß sie relativ wenig Speicherplatz benötigt, und bei geschickter Ausgabe der beschriebenen Flächen von hinten nach vorne erhalten wir automatisch beim Flächenmodell ein Unterdrücken der verdeckten Flächen (siehe Abbildung 7-14). Wir müssen also nicht wie bei der vorigen Struktur die Flächen erst vor der Ausgabe nach ihrer Tiefenlage sortieren. Angewendet habe ich diese Darstellungsweise zum Beispiel in dem Programm *apfel3d.c*, daß räumliche Darstellungen von Apfelmännchen-Ausschnitten generiert.

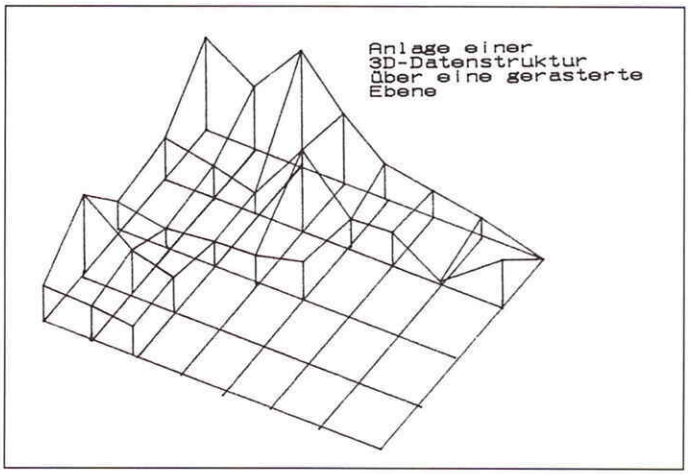
3) Pseudo 3-D-Grafik

Als letztes Beispiel möchte ich eine Methode vorstellen, die ohne jede Anlage einer Daten-Struktur auskommt. Sie ist im Prinzip ähnlich der Methode 2), arbeitet aber mit der Rasterung des Bildschirms und eignet sich vor allem zur Darstellung von Apfelmännchen und seinen Verwandten. Wir brauchen uns keine Gedanken über eine Projektion oder Hidden-Lines zu machen.

Wir tragen über die Pixel einer Bildschirmzeile die gewünschten Höhen auf. Ist eine Zeile fertig gezeichnet, beginnen wir wieder eine Zeile tiefer und einen oder mehrere Pixel früher (siehe Abbildung 7-16). Sind dann einige Linien gezeichnet, entsteht das räumliche Bild. Beim Zeichnen der Höhe `h` gehen wir folgendermaßen vor: Wir ziehen eine

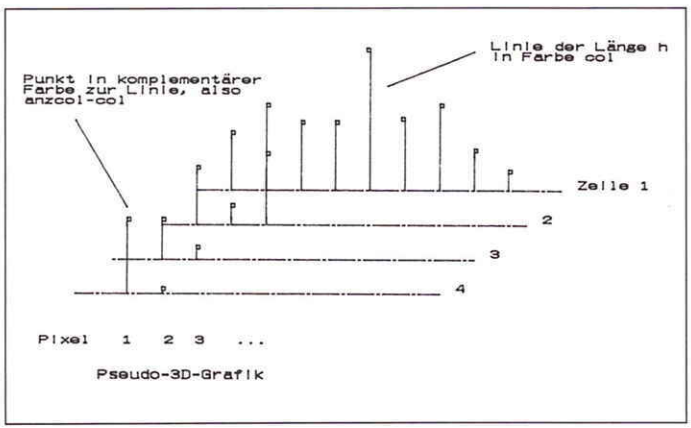


7-14 Beispiel eines Flächenmodells



7-15

Linie in der gewünschten Farbe vom augenblicklichen Zeilenpunkt (x,y) zum Punkt (x,y-h). Damit später der Unterschied zwischen senkrechter Wand und Hochplateau zu erkennen ist, setzen wir an (x,y-h) einen Punkt mit der zu unserer gewünschten (col) komplementären Farbe (sind anzcol Farbe verfügbar, so bedeutet dies: Der Punkt wird mit der Farbe (anzcol-col) gezeichnet). Verwendet habe ich dieses Verfahren zum Beispiel *im Programm fraktal.c*. Sollte jemand das Vorgehen nicht verstanden haben, so ist es anzuraten, einfach einmal bei der Generierung eines solchen Bildes zuzuschauen.



7-16

Mit dem folgenden Programm möchte ich diese kurze Einführung in die Grundlagen dreidimensionaler Grafik beenden. Dieses eigentlich nicht zum Thema des sonstigen Buches passende Kapitel wurde eingefügt, um die Berechnungen und das Vorgehen in einigen Programmen, in denen dreidimensionale Darstellungen gebraucht werden, besser verstehen zu können.

Ich habe im Programm aus Gründen des Umfangs auf eine Flächendarstellung der Körper und auf einen »double-buffered Display« verzichtet. Das »double-buffered Display« hätte den Vorteil, daß das Neuzeichnen eines Bildes nicht sichtbar vor sich geht und das fertig gemalte Bild erst dann eingeblendet wird (unter der Verwendung von der Routine LoadView()). Die Folge ist, daß das Bild dann auch bei bewegter Grafik nicht flimmert. Gerade das Arbeiten des Blitters beim Flächenfüllen ist sehr störend, so daß

bei der Ausgabe als Flächenmodell diese Display-Art verwendet werden sollte. Wen es genauer interessiert, wie ein Screen ohne Verwendung des Intuition geöffnet und bedient wird, der sollte einen Blick in das Programm *screen.c* werfen. Schreiben Sie doch einen Hidden-Lines-Algorithmus für die Ausgabe als Flächenmodell.

Das Programm verlangt beim Starten die Übergabe eines Namens. Es muß ein Definitions-File *name.dat* vorhanden sein, in dem die Punkt- und Flächendefinitionen bestimmt werden. Diese Vorgehensweise hat den Vorteil, daß auch ohne erneutes zeit- und speicherplatzaufwendiges Kompilieren problemlos neue Definitionen erfolgen können.

Ein Definitions-File ist eine einfache Textdatei, und hat folgenden Aufbau:

```
' '
```

Bedeutet eine REM-Zeile, deren weiterer Inhalt ignoriert wird.

```
'P' x,y,z
```

Gibt an, daß nun eine Punktdefinition erfolgt. Dann folgt die x-, y- und z-Koordinate (*float* Werte zwischen 0 und 1) mit »,« getrennt, also zum Beispiel `'P .1,.5,.9'`. Die Punkte werden nach der Reihenfolge ihrer Definition durchnumeriert.

```
'F' P1,P2,...Pn
```

definiert ein beliebiges n-Eck als Fläche. Es folgen die Punkte P1,P2,...Pn, die die Fläche begrenzen, also zum Beispiel `'F 1,2,3'` würde ein Dreieck bestehend aus den Punkten 1, 2 und 3 definieren.

```
'B' wert
```

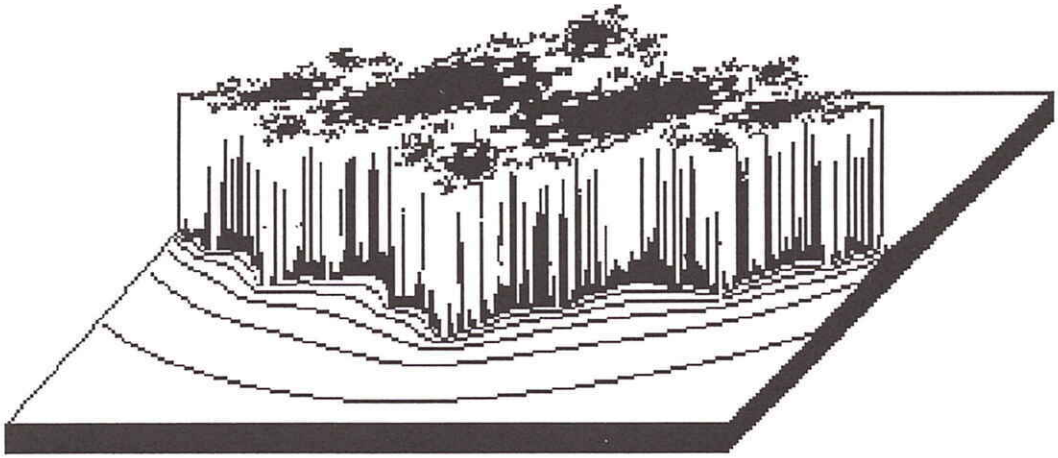
Gibt den Abstand zwischen Betrachter und Bildschirm an. *wert* ist ein *float* Wert. Ist er kleiner als 1, so erfolgt die Berechnung mit Perspektive, ansonsten erfolgt nur eine Projektion.

Werden bei der Eingabe Fehler gemacht, so meldet sich das Programm entsprechend.

Nachdem wir das Programm gestartet haben und das Definitions-File eingelesen werden konnte, erscheint das Gittermodell des eingegebenen Körpers auf dem Bildschirm. Nun können wir durch Drücken verschiedener Tasten Translationen, Drehungen und Skalierungsveränderungen vornehmen, und so die Wirkung der oben vorgestellten Transformationen überprüfen.

Folgende Tasten-Funktionen sind vorhanden:

- F1 Verschiebung des Objekts in positive x-Richtung.
- F2 Verschiebung des Objekts in negative x-Richtung.
- F3 Verschiebung des Objekts in positive y-Richtung.
- F4 Verschiebung des Objekts in negative y-Richtung.
- F5 Verschiebung des Objekts in positive z-Richtung.
- F6 Verschiebung des Objekts in negative z-Richtung.
- F9 Verkleinern.
- 1 Absolute Drehung um die x-Achse in pos. Richtung.
- 2 Absolute Drehung um die x-Achse in neg. Richtung.
- 3 Absolute Drehung um die y-Achse in pos. Richtung.
- 4 Absolute Drehung um die y-Achse in neg. Richtung.



7-17 Ergebnis der Pseudo-3-D-Grafik

- [5] Absolute Drehung um die z-Achse in pos. Richtung.
- [6] Absolute Drehung um die z-Achse in neg. Richtung.
- [7] Relative Drehung um die x-Achse in pos. Richtung.
- [8] Relative Drehung um die x-Achse in neg. Richtung.
- [9] Relative Drehung um die y-Achse in pos. Richtung.
- [Ø] Relative Drehung um die y-Achse in neg. Richtung.
- [-] Relative Drehung um die z-Achse in pos. Richtung.
- [=] Relative Drehung um die z-Achse in neg. Richtung.

Relativ bedeutet hier Drehung vor der Translation, also Drehung im Objektsystem. »absolut« bedeutet dagegen Drehung nach der Verschiebung.

- [R] Macht alle bisherigen Veränderungen ungeschehen.
- [E] beendet das Programm.
- [A] Animiert alle bisher eingegebenen Aktionen.
- [S] Speichern der Animation in einem »*datei.ani*« File.
- [L] Laden einer Animation (*datei.ani*).

Die Animation ist gerade bei einfachen Objekten recht schnell – ich war nach dem Fertigstellen des Programms selbst erstaunt. Alle Bewegungen, die über die Tastatur nach Starten beziehungsweise »Reset-en« des Programms vorgenommen wurden, werden in einer Tabelle abgelegt. Die Ablage erfolgt folgendermaßen:

Ausgabe-Feld

bx	by	(Bildschirm-Koordinaten)
-3		Bedeutet Endkennung für das Ausgabe-Feld. (Bei diesem Wertes wird die Ausgabe beendet.)
-2		Ein neues Bild fängt an. (Der Screen muß gelöscht werden.)
-1		Eine neue Fläche beginnt.

(Die letzten Punkte bilden eine Fläche, die jetzt gezeichnet werden kann. Auf `-1` folgt immer ein Wert, der die Farbe der Fläche angibt, damit auch diese nicht berechnet werden muß. Im augenblicklichen Zustand des Programms steht hier immer der gleiche Wert (blau), und es werden keine Flächen gezeichnet, sondern nur deren Begrenzungskanten!

Bei der Animation braucht also nichts mehr berechnet werden; Projektion usw. fällt damit weg – deshalb die recht hohe Geschwindigkeit.

Abschließend noch zwei Beispiele für Definitionsdateien:

```
*****************************
* Definition einer Pyramide *
*****************************

P .1,.1,-.1
P -.1,.1,-.1
P -.1,-.1,-.1
P .1,-.1,-.1
P Ø,Ø,.2

F 1,2,5
F 2,3,5
F 3,4,5
F 4,1,5
F 1,2,3,4

B 1
```

Als zweites Beispiel ein komplexeres Objekt – ein Flugzeug.

```
************
* Flugzeug *
************
```

Rumpf-Definition (Punkt 1–10)

```
P .1,Ø,.2
P .1,.1,.2
P -.1,Ø,.2
P -.1,.1,.2
P -.1,Ø,-.2
P -.1,.1,-.2
P .1,Ø,-.2
P .1,.1,-.2
P .Ø5,Ø,.8
P -.Ø5,Ø,.8

F 1,2,8,7
F 3,4,6,5
F 5,6,8,7
F 2,4,6,8
F 1,9,1Ø,3,5,7
F 1,2,9
F 3,4,1Ø
F 9,1Ø,4,2
```

Leitwerk-Definition (Punkt 11–15)

```
P .Ø2,.1,-.2
P -.Ø2,.1,-.2
P .Ø2,.3,-.2
P -.Ø2,.3,-.2
P Ø,.1,.18

F 11,12,14,13
F 11,12,15
F 13,14,15
F 15,12,13
```

Fluegell-Definition (Punkt 16–21)

```
P .1,.Ø3,.2
P .1,.Ø3,-.15
P .1,.Ø7,-.15
P .1,.Ø7,.2
```

```
P .6,.Ø5,.Ø5
P .6,.Ø5,-.Ø5

F 16,17,21,2Ø
F 18,19,2Ø,21
F 16,2Ø,19
F 17,21,18
```

Fluegel2-Definition (Punkt 22–27)

```
P -.1,.Ø3,.2
P -.1,.Ø3,-.15
P -.1,.Ø7,-.15
P -.1,.Ø7,.2
P -.6,.Ø5,.Ø5
P -.6,.Ø5,-.Ø5

F 22,23,24,25
F 24,25,26,27
F 22,26,25
F 23,24,27
```

Abstand Bild – Screen

```
B 1
```

Programm: *3-d.c*

```
 1 /*****************************************************************************/
 2 /* 3-d.c                                                        13.12.87 */
 3 /* Version 1.Ø                                                           */
 4 /*                                                                       */
 5 /*                                                                       */
 6 /* Beispielprogramm zur Demonstration dreidimensionaler Grafiken         */
 7 /*                                                                       */
 8 /* Das Programm wird unter Angabe einer Arbeitsdatei ('datei') gestartet. */
 9 /* Die Definition der Koerper erfolgt in einem File mit dem Namen       */
1Ø /* 'datei.dat', das nach dem Starten des Programms eingelesen wird. Sol- */
11 /* te dieses File nicht existieren, so bricht das Programm ab.          */
12 /*                                                                       */
13 /* Das Erstellen des 'datei.dat' Files erfolgt folgendermassen:         */
14 /* ('datei.dat' ist eine ASCII-Datei und kann mit dem Befehl 'ed'erstellt */
15 /* werden.)                                                             */
16 /* Beginnt eine Zeile mit einem Leerzeichen oder ist sie leer, so wird  */
17 /* diese igmoriert, also                                                */
18 /* ' ' ... bedeutet Rem-Zeile                                           */
19 /* 'P' x,y,z  definiert einen Punkt nach x-, y- und z-Koordinate. (x,y,z */
2Ø /*           float-Werte zwischen Ø und 1.) Die Punkte werden nach der  */
21 /*           Reihenfolge ihrer Definition numeriert. Der Punkt Ø ist    */
22 /*           also der erste Punkt, der in '.dat' File definiert wurde.  */
23 /* 'F' P1,P2,P3...Pn definiert eine Flaeche, die aus den Punkten P1, P2, */
24 /*           ...Pn begrenzt wird. Auch die Flaechen werden nach der     */
25 /*           Reihenfolge ihrer Definition numeriert.                    */
26 /* 'B' wert  'wert' gibt den Abstand zwischen Betrachter und Bildschirm */
27 /*           an und ist wichtig fuer die Perspektiveberechnung. 'wert'  */
28 /*           ist ein float-Wert. 'wert=1' bedeutet maximale Verzerrung. */
29 /*                                                                      */
3Ø /* Falls bei der Definition ein Fehler auftritt, wird dies gemeldet. Die */
31 /* Ueberpruefgung ist aber nicht 1ØØ%!!                                 */
32 /*                                                                      */
33 /* Das Programm wird ueber die Tastatur bedient.                        */
34 /* Taste:                                                               */
35 /* 'F1'   Verschiebung des Objekts in positive x-Richtung               */
36 /* 'F2'   Verschiebung des Objekts in negative x-Richtung               */
37 /* 'F3'   Verschiebung des Objekts in positive y-Richtung               */
38 /* 'F4'   Verschiebung des Objekts in negative y-Richtung               */
39 /* 'F5'   Verschiebung des Objekts in positive z-Richtung               */
4Ø /* 'F6'   Verschiebung des Objekts in negative z-Richtung               */
41 /* 'F9'   Verkleinern                                                   */
42 /* 'F1Ø'  Vergroessern                                                  */
43 /* '1'    absolute Drehung um die x-Achse in pos. Richtung              */
44 /* '2'    absolute Drehung um die x-Achse in neg. Richtung              */
45 /* '3'    absolute Drehung um die y-Achse in pos. Richtung              */
46 /* '4'    absolute Drehung um die y-Achse in neg. Richtung              */
47 /* '5'    absolute Drehung um die z-Achse in pos. Richtung              */
48 /* '6'    absolute Drehung um die z-Achse in neg. Richtung              */
49 /* '7'    relative Drehung um die x-Achse in pos. Richtung              */
5Ø /* '8'    relative Drehung um die x-Achse in neg. Richtung              */
51 /* '9'    relative Drehung um die y-Achse in pos. Richtung              */
52 /* 'Ø'    relative Drehung um die y-Achse in neg. Richtung              */
53 /* '-'    relative Drehung um die z-Achse in pos. Richtung              */
54 /* '='    relative Drehung um die z-Achse in neg. Richtung              */
```

```
/*       'relative'bedeutet Drehung vor der Translation, also Drehung im  */
/*       Objektsystem. 'absolut' bedeutet dagegen Drehung nach der Ver-   */
/*       schiebung.                                                       */
/* 'R'   macht alle bisherigen Drehungen, Verschiebungen und Vergroes-    */
/*       serungen ungeschehen.                                            */
/* 'E'   beendet das Programm                                             */
/* 'A'   animiert alle bisher eingegebenen Aktionen.                      */
/* 'S'   Speichern der Animation in einem 'datei.ani' File                */
/* 'L'   Laden einere Animation ('datei.ani')                             */
/*                                                                        */
/*                                                                        */
/* fuer Markt+Technik geschrieben von Heiko Knappe                        */
/*                                                                        */
/* 3D-Grafik                                                              */
/**************************************************************************/
/* Im Datenfile koennen Flaechen mit beliebig vielen Eckpunkten, also     */
/* Polygone definiert werden.                                             */


#include<stdio.h>
#include<math.h>
#include<exec/types.h>
#include<libraries/dos.h>
#include<intuition/intuition.h>

#define PI 3.14
#define ANZ 200
#define PANZ 500
#define FANZ 200
#define N 10000
#define WIDTH 320  /* Bildgroesse */
#define HEIGHT 200
#define DEPTH 5
#define WSTEP .1   /* Winkelzunahme */
#define STEP .05   /* Zunahme bei der Translation */
#define MUL .05    /* Vergroesserungsfaktor */

/* wichtige Struktur-Definitionen */
struct IntuitionBase *IntuitionBase;
struct GfxBase *GfxBase;
struct IntuiMessage *MyMessage;
struct Screen *MyScreen;
struct Window *MyWindow;
struct RastPort *rp;
struct NewWindow MyNewWindow=
 {0,0,WIDTH,HEIGHT,0,1,RAWKEY,ACTIVATENOCAREREFRESHBORDERLESS,0,0,0,
  0,0,0,0,0,0,0,CUSTOMSCREEN};
struct NewScreen MyNewScreen=
 {0,0,WIDTH,HEIGHT,DEPTH,0,1,0,CUSTOMSCREEN,0,0,0,0};

/* Variablen-Definitionen                    */
/* Definitionen des realen Koerpers          */
/* Punkt-Koordinaten (x,y,z) im Realsystem   */
float px[ANZ],py[ANZ],pz[ANZ];
/* nach der Transformation                   */
float pxh[ANZ],pyh[ANZ],pzh[ANZ];
/* Punktetabelle                             */
int p[PANZ];
/* Flaechentabelle, Hilfstabelle um Sortieren */
short f[FANZ],fh[FANZ];
/* Definition der Bildschirmkoordinaten-Tabelle */
short bx[N],by[N];
/* Zaehler */
int n,np,nf,i,j,k;
short anzcol;  /* Anzahl der Farben des Screens */
/* Perspektive Parameter                     */
float b=1;
/* Definition der Winkel und Transformationsparameter */
float winkelax=0,winkelay=0,winkelaz=0;
float winkelrx=0,winkelry=0,winkelrz=0;
float winkelp=30;
float transx=0,transy=0,transz=0;
/* Zaehler fuer die Bildschirmkoordinaten */
int count,countlast=0;
/* Vergroesserungfaktor */
float m=1;
/* Name des Bildes */
char name[128],name_[128];
```

```
/*********************/
/* Hauptprogramm     */
/*********************/

main(argc,argv)
int argc;
char *argv[];
{
 if(argc!=2)
 {
    printf("Eingabe: %s file.dat\n",argv[0]);
    close_();
 }
 strcpy(name,argv[1]);
 einlesen();
  open_();
 new();
 for(;;)
 {
    while((MyMessage=GetMsg(MyWindow->UserPort))==0);
    ReplyMsg(MyMessage);
    if((MyMessage->Class)==RAWKEY) do_key();
 }
}


/*********************/
/* Tastatureingabe   */
/* testen            */
/*********************/

do_key()
{
 short key;

 key=MyMessage->Code;
 switch(key)
 {
    case 1 : /* '1' Drehung um x-Achse in pos. Richtung */
             /* im Koordinatensystem */
             winkelax+=WSTEP;
             new();
             break;
    case 2 : /* '2' Drehung um x-Achse in neg. Richtung */
             /* im Koordinatensystem */
             winkelax-=WSTEP;
             new();
             break;
    case 3 : /* '3' Drehung um y-Achse in pos. Richtung */
             /* im Koordinatensystem */
             winkelay+=WSTEP;
             new();
             break;
    case 4 : /* '4' Drehung um y-Achse in neg. Richtung */
             /* im Koordinatensystem */
             winkelay-=WSTEP;
             new();
             break;
    case 5 : /* '5' Drehung um z-Achse in pos. Richtung */
             /* im Koordinatensystem */
             winkelaz+=WSTEP;
             new();
             break;
    case 6 : /* '6' Drehung um z-Achse in neg. Richtung */
             /* im Koordinatensystem */
             winkelaz-=WSTEP;
             new();
             break;
    case 7 : /* '7' Drehung um x-Achse in pos. Richtung */
             /* im Objektsystem */
             winkelrx+=WSTEP;
             new();
             break;
    case 8 : /* '8' Drehung um x-Achse in neg. Richtung */
             /* im Objektsystem */
             winkelrx-=WSTEP;
             new();
             break;
    case 9 : /* '9' Drehung um y-Achse in pos. Richtung */
             /* im Objektsystem */
             winkelry+=WSTEP;
```

```
            new();
            break;
    case 1Ø: /* 'Ø' Drehung um y-Achse in neg. Richtung */
            /* im Objektsystem                          */
            winkelry-=WSTEP;
            new();
            break;
    case 11: /* '?' ('-')  Drehung um z-Achse in pos. Richtung */
            /* im Objektsystem                                  */
            winkelrz+=WSTEP;
            new();
            break;
    case 12: /* ''' Drehung um z-Achse in neg. Richtung */
            /* im Objektsystem                          */
            winkelrz-=WSTEP;
            new();

            break;
    case 85: /* 'F6' Bewegung in pos. z-Richtung */
            transz+=STEP;
            new();
            break;
    case 84: /* 'F5' Bewegung in neg. z-Richtung */
            transz-=STEP;
            new();
            break;
    case 83: /* 'F4' Bewegung in pos. y-Richtung */
            transy+=STEP;
            new();
            break;
    case 82: /* 'F3' Bewegung in neg. y-Richtung */
            transy-=STEP;
            new();
            break;
    case 81: /* 'F2' Bewegung in pos. x-Richtung */
            transx+=STEP;
            new();
            break;
    case 8Ø: /* 'F1' Bewegung in neg. x-Richtung */
            transx-=STEP;
            new();
            break;
    case 88: /* 'F9' Vergroessern des Objekts */
            m+=MUL;
            new();
            break;
    case 89: /* 'F1Ø' Verkleinern des Objekts */
            if(m>Ø) m-=MUL;
            new();
            break;
    case 18: /* 'E' Ende */
            close_();
    case 19: /* 'R' Reset */
            winkelrx=winkelry=winkelrz=Ø;
            winkelax=winkelay=winkelaz=Ø;
            transx=transy=transz=Ø;
            count=countlast=Ø;
            m=1;
            new();
            break;
    case 32: /* 'A' Animation */
            ausgabe(Ø);
            break;
    case 33: /* 'S' Speichern der Animation */
            save_();
            break;
    case 4Ø: /* 'L' Laden der Animation */
            load_();
            break;
    default: /* keine belegte Taste gedrueckt */
            break;
 }
}


/*********************/
/* neues Bild        */
/* zeichnen          */
/*********************/

new()
{
 copy();
```

```
298   rotx(winkelrx);
299   roty(winkelry);
300   rotz(winkelrz);
301   trans(transx,transy,transz);
302   rotx(winkelax);
303   roty(winkelay);
304   rotz(winkelaz);
305   per(b);
306   transform(winkelp);
307   SetRast(rp,0);
308   ausgabe(countlast);
309  }
310
311
312  /*********************/
313  /* Initialisierung   */
314  /*********************/
315
316  open_()
317  {
318   if((IntuitionBase=OpenLibrary("intuition.library",0))==0)
319      close_();
320   if((GfxBase=OpenLibrary("graphics.library",0))==0)
321      close_();
322   if((MyScreen=OpenScreen(&MyNewScreen))==0)
323      close_();
324   MyNewWindow.Screen=MyScreen;
325   if((MyWindow=OpenWindow(&MyNewWindow))==0)
326      close_();
327   rp=MyWindow->RPort;
328   set_color();
329  }
330
331  close_()
332  {
333   if(IntuitionBase!=0) CloseLibrary(IntuitionBase);
334   if(GfxBase!=0) CloseLibrary(GfxBase);
335   if(MyWindow!=0) CloseWindow(MyWindow);
336   if(MyScreen!=0) CloseScreen(MyScreen);
337   exit(0);
338  }
```

```
339
340  set_color()
341  {
342   for(anzcol=1,i=0;i<DEPTH;++i,anzcol*=2);
343   for(i=0;i<anzcol;++i)
344      SetRGB4(&(MyScreen->ViewPort),i,0,0,(short)(15*(float)i/(float)anzcol));
345  }
346
347
348  /*********************/
349  /* Laden, Speichern  */
350  /*********************/
351
352  save_()
353  {
354   int file;
355
356   /* an den Bildnamen wird '.ani' angehaengt */
357   strcpy(name_,name);
358   strcpy(name_+strlen(name_),".ani");
359   file=Open(name_,MODE_NEWFILE);
360   if(file==0)
361   {
362      printf("Animations-File '%s' kann nicht geoeffnet werden!\n",name_);
363      close_();
364   }
365   Write(file,&count,4); Write(file,&countlast,4);
366   Write(file,&b,4); Write(file,&m,4);
367   Write(file,&n,4); Write(file,&np,4); Write(file,&nf,4);
368   Write(file,&transx,4); Write(file,&transy,4); Write(file,&transz,4);
369   Write(file,&winkelax,4); Write(file,&winkelay,4); Write(file,&winkelaz,4);
370   Write(file,&winkelrx,4); Write(file,&winkelry,4); Write(file,&winkelrz,4);
371   Write(file,&bx[0],2*count);
372   Write(file,&by[0],2*count);
373   Close(file);
374  }
375
376  load_()
377  {
378   int file;
379
```

```
38Ø  /* an den Bildnamen wird '.ani' angehaengt */
381  strcpy(name_,name);
382  strcpy(name_+strlen(name_),".ani");
383  file=Open(name_,MODE_OLDFILE);
384  if(file==Ø)
385  {
386     printf("Animations-File '%s' kann nicht geoeffnet werden!\n",name_);
387     close_();
388  }
389  Read(file,&count,4); Read(file,&countlast,4);
39Ø  Read(file,&b,4); Read(file,&m,4);
391  Read(file,&n,4); Read(file,&np,4); Read(file,&nf,4);
392  Read(file,&transx,4); Read(file,&transy,4); Read(file,&transz,4);
393  Read(file,&winkelax,4); Read(file,&winkelay,4); Read(file,&winkelaz,4);
394  Read(file,&winkelrx,4); Read(file,&winkelry,4); Read(file,&winkelrz,4);
395  Read(file,&bx[Ø],2*count);
396  Read(file,&by[Ø],2*count);
397  Close(file);
398  new();
399 }
4ØØ
4Ø1 /* Einlesen der Datei '.dat' */
4Ø2 einlesen()
4Ø3 {
4Ø4  int file;
4Ø5  short zeile=Ø;
4Ø6  char line[8Ø],*ii;
4Ø7
4Ø8  /* an den Bildnamen wird '.dat' angehaengt */
4Ø9  strcpy(name_,name);
41Ø  strcpy(name_+strlen(name_),".dat");
411  file=Open(name_,MODE_OLDFILE);
412  if(file==Ø)
413  {
414     printf("Daten-File '%s' kann nicht geoeffnet werden!\n",name_);
415     close_();
416  }
417  while(inline(line,file,8Ø)!=Ø)
418  {
419     ++zeile;
42Ø     switch(line[Ø])
421     {
422        case ' ': /* Rem-Zeile */
423                  break;
424        case 1Ø : /* Return-Zeile */
425                  break;
426        case 'P': /* Punktdefinition */
427                  if(sscanf(line+2,"%f,%f,%f",&px[n],&py[n],&pz[n])!=3)
428                     err(zeile,line);
429                  else
43Ø                     ++n;
431                  break;
432        case 'F': /* Flaechendefinition */
433                  i=1; j=Ø;
434                  while(sscanf(line+i,"%d",&p[np+j])==1)
435                  {
436                     ++j;
437                     ii=strchr(line+i,',');
438                     if(ii!=Ø) i=ii-line+1;
439                     else break;
44Ø                  }
441                  if(j<3) /* falsche Eingabe */
442                     err(zeile,line);
443                  else
444                  {
445                     /* Mit Punkt 'Ø' beginnen */
446                     for(i=Ø;i<j;--p[np+i],++i);
447                     f[nf]=np;
448                     ++nf;
449                     np+=j;
45Ø                     f[nf]=np;
451                  }
452                  break;
453        case 'B': /* Abstand Screen - Betrachter */
454                  if(sscanf(line+2,"%f",&b)!=1)
455                  {
456                     err(zeile,line);
457                     b=1;
458                  }
459                  break;
46Ø        default : /* Fehler aufgetreten */
```

```
461                 err(zeile,line);
462                 break;
463     }
464  }
465  /* falls nicht sortiert wird! */
466  for(i=Ø;i<(nf+1);fh[i]=i,++i);
467  Close(file);
468 }
469
47Ø err(z,l) /* Fehler? */
471 int  z;
472 char *l;
473 {
474  printf("Fehler in Zeile %d:\n  -> %s",z,l);
475 }
476
477 inline(c,handle,len) /* eine Zeile des Files einlesen */
478 char c[];
479 long *handle;
48Ø short len;
481 {
482  short i=Ø,ende=Ø;
483
484  while((i<len)&&(c[i-1]!=1Ø)&&((ende=Read(handle,&c[i],1))!=Ø))
485     ++i;
486  c[i]=Ø;
487  return(ende);
488 }
489
49Ø
491 /*********************/
492 /* Ausgaberoutine    */
493 /*********************/
494
495 ausgabe(counter)
496 int counter;
497 {
498  short xh,yh;
499  char col;
5ØØ
```

```
5Ø1  /* bx[]=-3 bedeutet Endkennung des Feldes fuer die Ausgabe */
5Ø2  for(;(counter<N)&&(bx[counter]!=-3);++counter)
5Ø3  {
5Ø4     /* faengt ein neues Bild an? (bx[]=-2) */
5Ø5     if(bx[counter]==-2)
5Ø6     {
5Ø7        ++counter;
5Ø8        SetRast(rp,Ø);
5Ø9     }
51Ø     /* faengt eine neue Flaeche an? (bx[]=-1) */
511     if(bx[counter]==-1)
512     {
513        ++counter;
514        if(bx[counter-2]!=-2) /* falls nicht erste Flaeche */
515           Draw(rp,xh,yh);    /* alte Flaeche schliessen   */
516        col=bx[counter++];
517        SetAPen(rp,col);
518        xh=bx[counter];
519        yh=by[counter];
52Ø        Move(rp,xh,yh);
521     }
522     else
523        Draw(rp,bx[counter],by[counter]);
524  }
525  Draw(rp,xh,yh);  /* letzte Flaeche schliessen   */
526 }
527
528
529
53Ø /*********************/
531 /* Transformations-  */
532 /* routinen          */
533 /*********************/
534
535 /* alle Transformationen finden in Hilfskoordinaten statt! */
536 rotx(wx)     /* Drehung um die x-Achse */
537 float wx;
538 {
539  float sinwx,coswx,yh,zh;
54Ø
```

```
541  sinwx=sin(wx);
542  coswx=cos(wx);
543  for(i=Ø;i<n;++i)      /* alle Punkte */
544  {
545     yh=pyh[i]*coswx-pzh[i]*sinwx;
546     zh=pyh[i]*sinwx+pzh[i]*coswx;
547     pyh[i]=yh;
548     pzh[i]=zh;
549  }
55Ø }
551
552 roty(wy)     /* Drehung um die y-Achse */
553 float wy;
554 {
555  float sinwy,coswy,xh,zh;
556
557  sinwy=sin(wy);
558  coswy=cos(wy);
559  for(i=Ø;i<n;++i)      /* alle Punkte */
56Ø  {
561     xh=pxh[i]*coswy+pzh[i]*sinwy;
562     zh=-pxh[i]*sinwy+pzh[i]*coswy;
563     pxh[i]=xh;
564     pzh[i]=zh;
565  }
566 }
567
568 rotz(wz)     /* Drehung um die z-Achse */
569 float wz;
57Ø {
571  float sinwz,coswz,xh,yh;
572
573  sinwz=sin(wz);
574  coswz=cos(wz);
575  for(i=Ø;i<n;++i)      /* alle Punkte */
576  {
577     xh=pxh[i]*coswz-pyh[i]*sinwz;
578     yh=pxh[i]*sinwz+pyh[i]*coswz;
579     pxh[i]=xh;
58Ø     pyh[i]=yh;
581  }
582 }
```

```
583
584 trans(x,y,z) /* Translation um (x,y,z) */
585 float x,y,z;
586 {
587  for(i=Ø;i<n;++i)      /* alle Punkte */
588  {
589     pxh[i]=pxh[i]+x;
59Ø     pyh[i]=pyh[i]+y;
591     pzh[i]=pzh[i]+z;
592  }
593 }
594
595 per(d)     /* Perspektive berechnen  */
596 float d;
597 {
598  float fak;
599
6ØØ  /* Perspektive wird nur fuer b<1 beruecksichtigt, sonst Projektion */
6Ø1  /* mit Hilfe der z-Koordinate                                      */
6Ø2  if(d>=1) return(Ø);
6Ø3  for(i=Ø;i<n;++i)      /* alle Punkte */
6Ø4  {
6Ø5     if(pzh[i]>d/2) fak=2;
6Ø6     else fak=d/(d-pzh[i]);
6Ø7     pxh[i]=fak*pxh[i];
6Ø8     pyh[i]=fak*pyh[i];
6Ø9     pzh[i]=Ø;
61Ø  }
611 }
612
613 copy()     /* Original-Daten kopieren */
614 {
615  for(i=Ø;i<n;++i)
616  {
617     pxh[i]=px[i]*m;
618     pyh[i]=py[i]*m;
619     pzh[i]=pz[i]*m;
62Ø  }
621 }
622
```

```
transform(wp) /* Bildpunkte berechnen */
float wp;     /* Projektionswinkel   */
{
 float coswp,sinwp;

 if(count>=(N-(f[nf]+2*nf+1)))
    return(Ø);
 countlast=count;
 coswp=cos(wp);
 sinwp=sin(wp);
 for(i=Ø;i<n;++i)
 {
   pxh[i]=pxh[i]-pzh[i]*s inwp;
   pyh[i]=pyh[i]-pzh[i]*c oswp;
 }
 bx[count++]=-2;          /* neues Bild   */
 for(i=Ø;i<nf;++i)
 {
    bx[count++]=-1;       /* neue Flaeche */
    bx[count++]=anzcol-1; /* Farbe        */
    for(j=f[fh[i]];j<f[fh[i]+1];++j)
    {
        bx[count]=(short)(HEIGHT/2*(1+pxh[p[j]]));
        by[count]=(short)(HEIGHT/2*(1-pyh[p[j]]));
        /* 'Clipping' */
        if(bx[count]<Ø) bx[count]=Ø;
        if(bx[count]>WIDTH) bx[count]=WIDTH-1;
        if(by[count]<Ø) by[count]=Ø;
        if(by[count]>HEIGHT) by[count]=HEIGHT-1;
        ++count;
    }
 }
 bx[count]=-3;      /* Endkennung   */
}
```

8 C auf dem Amiga

8.1 Intuition - oder ein Computer zum Anfassen

Intuition, die Benutzeroberfläche des Amiga, stellt die Schnittstelle zwischen Mensch und Rechner her. Eine solche Kontaktstelle gut zu realisieren ist sicher eine sehr große Herausforderung. Beim Amiga ist sie, wie ich finde, gut gelungen, denn die Zusammenarbeit von Multitasking, Screens, Windows, Menüs und Gadgets ist ziemlich perfekt gelöst. Und das wichtigste: Das Betriebssystem unterstützt mit vielen Routinen die Intitialisierung und Handhabung dieser Elemente, so daß wir sie selbst einfach und ohne großen Aufwand benutzen können.

Bei der Arbeit mit dem Amiga kommen wir also nicht um Intuition herum. Da die Entwickler des Amiga ihre gesamte Dokumentation in erster Linie in C geschrieben haben, kommen wir auch nicht umhin, C zu programmieren. Die C-Programmierung ist mittlerweile schon recht gut (auch in deutsch) dokumentiert. Deshalb soll an dieser Stelle weder ein C-Handbuch, noch ein Handbuch über die Programmierung des Amiga stehen. Aber ich glaube, es erspart oft viel Zeit und Geld, nicht in einer Unmenge anderer Bücher nachblättern zu müssen. Dies Kapitel soll die Routinen, die in den Programmen des Buches verwendet werden, kurz (auch an kleinen Beispielen) vorstellen und mühsames Nachschlagen vermeiden. Es ist ein Extrakt aus der Fertigkeit, die nötig ist, den Amiga ein wenig zu beherrschen und wird dem Anfänger sicher hilfreich sein; will jemand tiefer eindringen, so ist das Studium der am Ende dieses Buches angegebenen Literatur sicher unvermeidbar. Doch nun zum ersten und wichtigsten Element der Intuition: den Screens.

8.1.1 Screens

Die Screens bilden das Fundament des Amiga-Grafiksystems und sind die Grundlage für Windows, Menüs, Gadgets, eigentlich für fast alles. Sie bestimmen die gewählte Auflösung und die Farben des Bildes. Ohne Screens geht es also nicht! Bei der Initialisierung eines Screens sind zuerst einmal die gewünschte Auflösung und der Anzeigemodus maßgebend für das Erscheinungsbild des Screens.

Es gibt vier verschiedene Auflösungen:

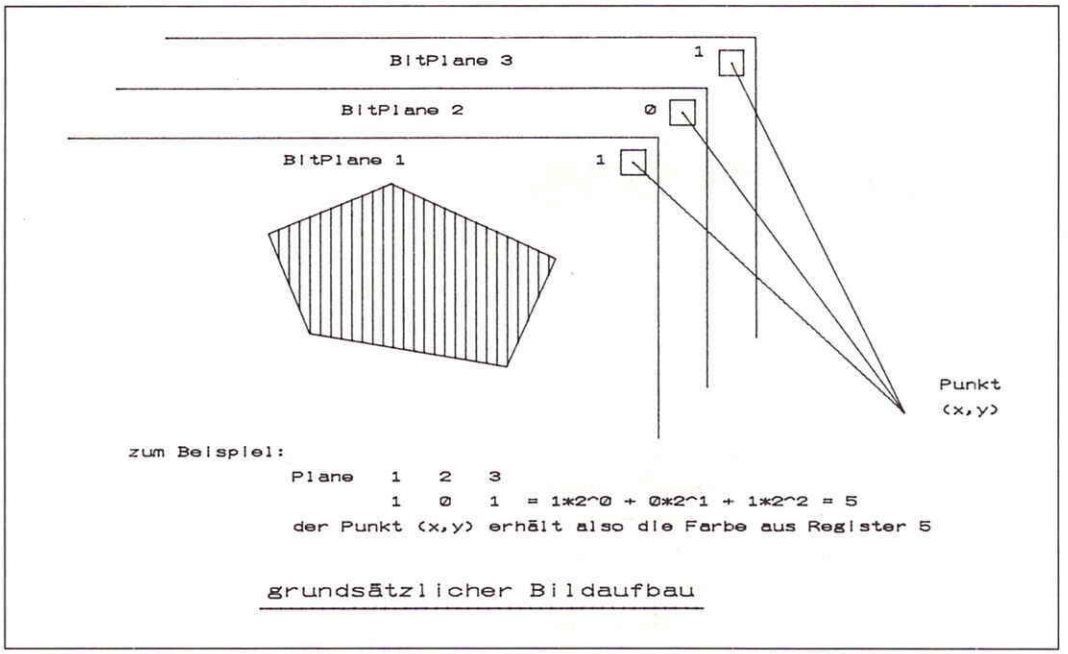
Niedrige Auflösung:
- 320 x 200 (250 PAL)
- max. 5 BitPlanes (32 Farben)

Hohe Auflösung:
- 640 x 200 (250 PAL)
- max. 4 BitPlanes (16 Farben)

Niedrige Auflösung / Interlace:
- 320 x 400 (500 PAL)
- max. 5 BitPlanes (32 Farben)

Hohe Auflösung / Interlace:
- 640 x 400 (500 PAL)
- max. 4 BitPlanes (16 Farben)

8-1



grundsätzlicher Bildaufbau

Aber damit nicht genug: Die verschiedenen Anzeigemodi machen den Amiga erst zum Grafiksystem:

- der »normale« Screen,
- der Hold-and-Modify Modus (4096 Farben gleichzeitig),
- der Extra-Half-Brite Modus (64 Farben gleichzeitig).

Bei den letzten beiden Modi ist zu beachten, daß ihre Farben zwar gleichzeitig auf einem Screen darstellbar sind, aber die Farbwahl gerade bei HAM (Hold-and-Modify) großen Einschränkungen unterworfen ist.

Zum besseren Verständnis noch einige Worte zum Aufbau eines Screens. Ein Screen besteht aus mehreren BitPlanes, wobei die gesetzten Bits die Wahl des Farbregisters und damit die Farbe eines Punktes bestimmen. Also zum Beispiel bei einem Screen mit drei Planes

```
            Plane    1  2  3
Punkt(1,1)           1  0  1
```

erhält dieser Punkt die Farbe des Farbregisters $1*2^0+0*2^1+1*2^2=5$ (siehe Abbildung 8-1).

Bei einem »normalen« Screen sind maximal fünf BitPlanes also 32 Farben möglich. Beim HAM-Modus wird nun eine sechste BitPlane verwendet. Die 16 unteren Farbregister, also Plane 1 bis 4, sind frei wählbar.

Die oberen zwei BitPlanes entscheiden, ob das Farbregister direkt geladen wird oder der Farbwert des links daneben liegenden Punktes modifiziert wird.

```
Plane   1  2  3  4  5  6
        1  1  0  0  0  0
```

bedeutet für diesen Punkt also, daß er die Farbe des Farbregisters 3 erhält.

Sind nun Bits der 5. oder 6. Plane gesetzt, so wird der `r`-, `g`- oder `b`-Anteil des Nach-

barpunktes neu bestimmt. Den Wert (von 0 bis 15) den dieser Anteil erhält wird dabei durch die unteren vier Planes angegeben.

```
Plane   5  6
        0  0   keine Modifizierung
        1  0  b-Wert wird modifiziert
        0  1  r-Wert wird modifiziert
        1  1  g-Wert wird modifiziert
```

Zum Beispiel bedeutet also

```
     Plane   1  2  3  4   5  6
Punkt(1Ø,1)  Ø  Ø  Ø  1   Ø  1
```

daß die Farbe des Punktes (9,1) übernommen wird und ihr r-Anteil den Wert 8 erhält.

Beim Extra-Halfbrite-Modus wird ebenfalls mit sechs BitPlanes gearbeitet. (Er ist nur auf europäischen PAL-Amigas möglich!) Die unteren 32 Farbregister (0–31) sind frei wählbar. Wird ein Farbregister von 32 bis 63 gewählt, so bedeutet dies, daß der Punkt die Farbe des Partnerregisters unter 31 mit halber Helligkeit erhält. Register 60 bedeutet also, daß die Farbe des Registers 60–31=29 mit halber Helligkeit verwendet wird. Es sind also 64 Farben möglich. Im Vergleich zum HAM-Modus kann jedoch die Farbe eines jeden Punktes direkt aus diesen 64 Farben gewählt werden und braucht nicht zum Erreichen einer bestimmten Farbe mehrere Schritte (maximal 3). Wie eben so oft erkauft man sich Vorteile mit Einschränkungen.

Einen Screen zu öffnen, ist eigentlich ganz einfach: Die Parameter eines Screens werden in der sogenannten *NewScreen*-Struktur festgelegt und dann eine Initialisierungsroutine `OpenScreen()` aufgerufen. (Für spezielle Anwendungen kann es aber manchmal auch ganz sinnvoll sein, `OpenScreen()` die Arbeit abzunehmen und alle notwendigen Initialisierungen selbst vorzunehmen. Dies war zum Beispiel bei dem »Filmprojektor« *pro.c* nötig.)

Die NewScreen-Struktur

```
struct NewScreen
{
SHORT  LeftEdge,TopEdge;
SHORT  Width,Height,Depth;
UBYTE  DetailPen,BlockPen;
USHORT ViewModes;
USHORT Type;
struct TextAttr *Font;
UBYTE  *DefaultTitle;
struct Gadget *FirstGadget;
struct BitMap *CustomBitMap;
}
```

`LeftEdge,TopEdge`

Geben die (x,y)-Koordinaten der linken oberen Screen-Ecke bezüglich des Bildschirms an. `LeftEdge` kann zwar angegeben werden, wird aber beim Öffnen ignoriert, da verschiedene Auflösungen nicht nebeneinander darstellbar sind.

`Width,Height,Depth`

In ihrer Reihenfolge geben diese Werte die Breite, Höhe (in Punkten) und die Anzahl der BitPlanes des Screens an.

Die Anzahl der BitPlanes ist maßgebend für die Farben, die in diesem Screen verwendet werden können (Anzahl der Farben = 2 *Depth*).

Wieviele BitPlanes ein Screen haben kann und wie groß er maximal sein darf, hängt von der verwendeten Auflösung ab.

Niedrige Auflösung:
- 320x200 (250 PAL)
- max. 5 BitPlanes (32 Farben)

Hohe Auflösung:
- 640x200 (250 PAL)
- max. 4 BitPlanes (16 Farben)

Niedrige Auflösung / Interlace:
- 320x400 (500 PAL)
- max. 5 BitPlanes (32 Farben)

Hohe Auflösung / Interlace:
- 640x400 (500 PAL)
- max. 4 BitPlanes (16 Farben)

Die Wahl der Auflösung erfolgt durch das Setzen von den Flags HIRES oder LACE in ViewModes.

`DetailPen,BlockPen`

Bestimmen die Farben mit denen die Gadgets und der Titeltext, bzw. die gefüllten Flächen gezeichnet werden sollen.

`ViewModes`

Dieser Wert wählt den Anzeigemodus des Screens. Er setzt sich aus folgenden möglichen Flags zusammen:

`HIRES`	hohe Auflösung
`LACE`	Interlace
`HAM`	Hold-and-Modify-Modus
`EXTRA_HALFBRITE`	Extra-Half-Brite-Modus
`(SPRITE`	der Screen enthält Sprites)
`(DUALPF`	Dual-Playfield Darstellung

`Type`

Soll eine anderweitig definierte BitMap-Struktur verwendet werden, so muß hier `CUSTOMBITMAP` stehen. Normalerweise steht hier aber `CUSTOMSCREEN`, dann initialisiert die Routine `OpenScreen()` selbst die notwendigen BitPlanes und die BitMap-Struktur.

`*Font`

Zeigt auf eine TextAttr-Struktur, in der ein besonderer Schriftsatz definiert wird, der in diesem Screen verwendet werden soll.

`*ScreenTitle`

Zeigt auf einen mit Null abgeschlossenen TextString, der den Namen des Screens angibt. Soll im Screen kein Name erscheinen, so muß hier eine Null stehen.

`*FirstGadget`

Zeigt auf die erste Gadget-Struktur der Gadget-Liste.

`*CustomBitMap`

Zeigt auf eine BitMap-Struktur. Dieser Wert wird nur dann berücksichtigt, wenn unter `Type` das CUSTOMBITMAP-Flag gesetzt ist.

Screen-Funktionen

Zur Vereinfachung werden folgende Variablen definiert:

```
struct NewScreen MyNewScreen;
struct Screen *MyScreen;
```

`MyNewScreen` ist also die Struktur selbst, wogegen `MyScreen` auf eine Screen-Struktur zeigt. Um die Adresse der MyNewScreen-Struktur im Speicher zu erhalten muß der Adress-Operator »&« verwendet werden.

Unter den Funktionen stehen die Register in denen die Aufrufparameter übergeben werden müssen. Diese muß ein C-Programmierer jedoch nicht beachten, denn für die richtige Übergabe sorgt schon der Compiler. Sie sind nur für die Programmierung in Assembler von Bedeutung und sind hier nur vollständigkeitshalber erwähnt.

```
MyScreen = OpenScreen(&MyNewScreen)
                              AØ
```

Öffnet einen Screen, dessen Parameter in der MyNewScreen-Struktur definiert sind. Diese Funktion übergibt einen Zeiger auf die Screen-Struktur des neuen Screens, falls das Öffnen erfolgreich war, andernfalls wird eine Null übergeben. Die Adresse der Screen-Struktur muß `MyScreen` zugewiesen werden, damit der Screen wieder mit dem Aufruf

```
CloseScreen(MyScreen)
                AØ
```

geschlossen werden kann. Es muß darauf geachtet werden, daß `MyScreen` wirklich auf eine Screen-Struktur zeigt (und nicht beispielsweise auf Null, da der Screen nicht geöffnet werden konnte), sonst führt der Aufruf zum Systemabsturz. Es wird kein Ergebnis übergeben.

Ein kurzes Programm zum Öffnen eines Screens könnte also wie folgt aussehen (ohne Definition der Screen Parameter):

```
struct NewScreen MyNewScreen = {...    };
struct Screen *MyScreen;

main()
{
 MyScreen=OpenScreen(&MyNewScreen)
 if(MyScreen==Ø) /* Öffnen erfolgreich */
    exit(FALSE);

 ...

 /* Überprüfung zwar unnötig, schadet aber nie! */
 if(MyScreen!=Ø) CloseScreen(MyScreen);
}
```

Die nun folgenden Funktionen »manipulieren« die Tiefenlage und Position von Screens.

Das eigentliche Aussehen eines Screens ist nach seinem Öffnen fest und kann nicht so ohne weiteres geändert werden. Zum Beispiel ist es nicht möglich, die Auflösung nachträglich zu ändern. Wird dies notwendig, so muß ein neuer Screen geöffnet werden (siehe Programm *fractal.c*).

```
ScreenToFront(MyScreen);
ScreenToBack (MyScreen);
                AØ
```

Legen den Screen auf dessen Screen-Struktur `MyScreen` zeigt vor bzw. hinter alle anderen geöffneten Screens. Falls mehrere Screens geöffnet worden sind, ist es am besten, man stellt sie sich als Papierblätter vor, die übereinander gelegt werden. Ein Nachhintenlegen bedeutet also, das erste Blatt Papier zu nehmen und hinter alle anderen zu legen.

Die Befehle

```
WBenchToFront();
```

und

```
WBenchToBack();
```

machen die Tiefenveränderung des Work-Bench-Screens möglich. Der Aufruf kommt dem gleichzeitigem Drücken der linken »Amiga«-Taste und »M« bzw. »N« gleich.

Der Workbench-Screen kann nur einmal existieren. Dieser Screen ist der Screen des Amigas, auf dem alle System-Ausgaben stattfinden. Die Bedeutung dieses Screens kennt jeder, der schon mit dem Amiga gearbeitet hat. Sollte es zum Beispiel aus Speicherplatzgründen notwendig werden, diesen zu schließen bzw. wieder zu öffnen, so geschieht dies mit den beiden folgenden Funktionen:

```
suc = CloseWorkBench();
suc = OpenWorkBench();
```

(suc ist entweder TRUE oder FALSE, je nachdem ob der Aufruf erfolgreich war.)

```
MoveScreen(MyScreen,dx,dy)
                A0    D0 D1
```

Verschiebt einen Screen um dy-Punkte nach unten, wenn dy größer als Null ist, ansonsten nach oben. Über die oberste Bildkante kann der Screen natürlich nicht gezogen werden. Der Wert dx kann angegeben werden, wird jedoch nicht berücksichtigt, da, wie schon angesprochen, verschiedene Screens nur untereinander und nicht nebeneinander darstellbar sind.

```
DisplayBeep(MyScreen);
                A0
```

Läßt den Screen kurz aufblitzen und bietet eine sehr einfache Möglichkeit, aufgetretene Fehler bemerkbar zu machen. Wird diese Funktion DisplayBeep(0); aufgerufen, so läßt Intuition alle geöffneten Screens blitzen.

Beispielprogramm: *openScreen.c*

```
/*****************************************************************************/
/* Oeffnen eines Screens                                                     */
/*                                                                           */
/* fuer Markt&Technik von H. Knappe                                          */
/*****************************************************************************/


#include <exec/types.h>
#include <intuition/intuition.h>

struct IntuitionBase *IntuitionBase;
struct GfxBase *GfxBase;

/* Nun wird die NewScreen-Struktur definiert                                 */
struct NewScreen MyNewScreen=
{
 0,                  /* Koordinate der linken Ecke                           */
 0,                  /* Koordinate der oberen Ecke                           */
 640,                /* Breite (high-resolution)                             */
 200,                /* Höhe (non-interlace)                                 */
 2,                  /* Tiefe (2 BitPlanes -> 16 Farben)                     */
 0,1,                /* DetailPen, BlockPen                                  */
 HIRES,              /* Anzeigemodus: high-resolution                        */
 CUSTOMSCREEN,       /* Screentyp                                            */
 NULL,               /* kein eigener Schriftsatz                             */
 "Mein erster Screen", /* Name des Screens                                   */
 NULL,               /* keine ScreenGadgets                                  */
 NULL                /* keine CustomBitMap                                   */
};



main()
{
 struct Screen *MyScreen;        /* Pointer auf die Screen-Struktur          */
 long i;

 printf("Oeffnen eines neuen Screens...\n");

 /* Oeffnen des IntuitionLibrary. Wenn das Ergebnis  gleich  Null ist,       */
 /* kann das IntuitionLibrary nicht geoeffnet werden und das  Programm       */
 /* sollte abbrechen.                                                        */
 IntuitionBase=(struct IntuitionBase*) OpenLibrary("intuition.library",0);
 if(IntuitionBase==NULL)
 {
    printf("Intuition.library kann nicht geoeffnet werden!\n");
    exit(FALSE);
 }
 printf("Intuition.library offen -> ");

 /* Oeffnen des  GraphicLibrary.  Wenn das Ergebnis  gleich  Null ist,       */
 /* kann das IntuitionLibrary nicht geoeffnet werden und das  Programm       */
 /* sollte abbrechen.                                                        */
 GfxBase=(struct GfxBase*)OpenLibrary("graphics.library",0);
 if(GfxBase==NULL)
 {
    printf("Graphic.library kann nicht geoeffnet werden!\n");
    CloseLibrary(IntuitionBase);
    exit(FALSE);
```

```
60  }
61  printf("Graphic.library offen -> ");
62
63  /* Oeffnen des Screens. Wenn das Ergebnis gleich  Null ist, konnte der     */
64  /* Screen nicht geoeffnet werden und das Programm sollte abbrechen!        */
65  if((MyScreen=(struct Screen*)OpenScreen(&MyNewScreen))==NULL)
66  {
67     printf("Screen kann nicht geoeffnet werden!\n");
68     CloseLibrary(GfxBase);
69     CloseLibrary(IntuitionBase);
70     exit(FALSE);
71  }
72  printf("Screen offen.\n");
73
74  /* Warteschleife */
75  for(i=1;i<=1000000;++i);
76
77  /* Programm beenden. */
78  CloseScreen(MyScreen);              /* Schliessen des Screens          */
79  printf("Screen geschlossen.\n");
80  exit(TRUE);
81 }
```

8.1.2 Fenster und Intuitionmeldungen

Sie sind das nächste, sehr wichtige Element des Intuition. Sie werden einem Screen zugeordnet und es können dort mehrere gleichzeitig vorhanden sein. Sie haben deshalb eine große Bedeutung, weil sie die »Nachrichtenempfänger« des Intuition sind. Um dies zu verstehen, müssen wir uns ein wenig mit dem Multitasking beschäftigen. Um zwischen unabhängigen Tasks Nachrichten übermitteln zu können, werden Ein-/Ausgabe-Kanäle eingerichtet, über die Informationen geschickt werden. Jedes Fenster erhält einen solchen Kanal nach dem Öffnen automatisch zugewiesen, so daß wir uns um die Initialisierung eines solchen Kanals nicht kümmern müssen. Welche Informationen, also welche Eingabeereignisse Intuition melden soll, wird mit den IDCMP-Flags *(Intuition Direct Communication Port)* festgelegt. Die Feststellung, ob eine Meldung vorliegt, und die eigentliche Übernahme geschieht durch Aufruf von `GetMsg()`. Vor einer neuen Abfrage muß, sofern eine Meldung übernommen wurde, diese mit `ReplyMsg()` quittiert werden. Geschieht dies nicht, ist ein Absturz sicher. Auch vor einem Verlassen des Programms sind alle Meldungen zu quittieren, da sonst die Meldung im Speicher nicht gelöscht wird. Und noch etwas: Ein Aufruf von `ReplyMsg()`, obwohl keine Meldung vorhanden ist, führt ebenfalls zum sicheren Absturz. Für uns wird also eine Abfrage folgendermaßen aussehen:

```
struct IntuiMessage *MyMessage;
struct Window *MyWindow;
...
MyMessage=GetMsg(MyWindow->UserPort);
if(MyMessage==0)
   ... /* keine Meldung */
else
   ReplyMsg(MyMessage);
```

`MyMessage` zeigt auf eine *IntuiMessage*-Struktur, die die Informationen über das Eingabeereignis enthält. Der `UserPort` des Fensters ist eine Art Briefkasten, in der von Intuition hinterlegt wird, ob ein relevantes Eingabeereignis vorgelegen hat. Um nun festzustellen, was geschehen war, muß die *IntuiMessage*-Struktur untersucht werden. Sie hat folgenden Aufbau:

```
struct IntuiMessage
{
struct Message ExecMessage;
ULONG  Class;
USHORT Code;
USHORT Qualifier;
APTR   IAddress;
SHORT  MouseX,MouseY;
ULONG  Seconds,Micros;
```

```
struct Window *IDCMPWindow;
struct IntuiMessage *SpecialLink;
}
```

Für uns sind hier nur zwei Struktur-Elemente interessant:

`Class`

enthält die gesetzten IDCMP-Flags, je nachdem welches Ereignis eingetreten ist.

`Code`

Beinhaltet einen Wert, der Zusatzinformationen je nach Ereignis gibt. Bei `RAWKEY` steht hier der Tastencode der gedrückten oder losgelassenen Taste, bei `MENUPICK` die Kennung des angewählten Menüpunkts und bei `GADGETUP/GADGETDOWN` die Kennung des Gadgets.

Mit der Überprüfung dieser beiden Werte können wir nun also feststellen, welches Ereignis eingetreten ist und es gleichzeitig auswerten. Eine typische Abfrage könnte also wie folgt aussehen:

```
struct Window *MyWindow;
struct IntuiMessage *MyMessage;
...
/* Warten bis etwas gemeldet wird */
while((MyMessage=GetMsg(MyWindow->UserPort))==Ø);
ReplyMsg(MyMessage); /* Meldung quittieren */
switch(MyMessage->Class)
{
   case CLOSEWINDOW: ...
   case RAWKEY: ...
   ...
}
```

Die Initialisierung eines Fensters verläuft genauso wie bei den Screens. Auch hier gibt es eine Struktur in der alle bestimmenden Parameter festgelegt werden, die *NewWindow*-Struktur.

Die NewWindow-Struktur

```
struct NewWindow
{
SHORT  LeftEdge,TopEdge;
SHORT  Width,Height;
UBYTE  DetailPen,BlockPen;
ULONG  IDCMP_Flags;
ULONG  Flags;
struct Gadget *FirstGadget;
struct Image *CheckMark;
UBYTE  *Title;
struct Screen *Screen;
struct BitMap *BitMap;
SHORT  MinWidth,MinHeight,MaxWidth,MaxHeight;
USHORT Type;
}
```

`LeftEdge,TopEdge`

Bestimmen die (x,y)-Position der linken oberen Ecke des neuen Windows relativ zum Screen, in dem es erscheint. Diese Koordinaten sind in Bildpunkten anzugeben.

`Width,Height`

Geben die Breite und die Höhe des Fensters an. Auch diese Werte sind in Bildpunkten anzugeben. Die »Tiefe«, das heißt die Anzahl der Farben, die im Fenster verwendet werden können, wird durch den Screen bestimmt, in dem es erscheint. Ein Window ist also dem Screen, in dem es erscheint, untergeordnet.

`DetailPen,BlockPen`

Farbregister für die Gadgets, Titeltext und Titelleiste.

`IDCMP_Flags`

Dieser Wert setzt sich auch verschiedenen Flags zusammen, die angeben, welche Ereig-

nisse dem Window vom Intuition gemeldet werden sollen. Wie das zu verstehen ist und welche Flags möglich sind, wird im nächsten Abschnitt erläutert.

`Flags`

Jeder weiß, wie unterschiedlich ein Fenster aussehen kann, manchmal ist es gar nicht mehr als ein solches zu erkennen. Um das Aussehen zu definieren, können folgende Flags wahlweise gesetzt werden:

`WINDOWCLOSE`	Fenster besitzt ein Close-Gadget.
`WINDOWDEPTH`	Die Tiefenlage kann verändert werden.
`WINDOWSIZING`	Die Größe kann verändert werden.
`WINDOWDRAG`	Das Fenster kann verschoben werden.
`BORDERLESS`	Das Fenster besitzt eine Umrandung (Border).
`ACTIVATE`	Das Fenster ist nach dem Öffnen aktiviert, das heißt, Intuition meldet nun diesem Fenster.
`GIMMEZEROZERO`	Der Punkt (`0,0`) ist nicht gleich der oberen linken Ecke, sondern der erste sichtbare Punkt im Fenster. Damit ist sichergestellt, daß nicht in der Titelleiste gezeichnet werden kann! Von den folgenden Flags muß eines gesetzt werden!
`SIMPLE_REFRESH`	Intuition meldet nach jeder Aktion am Fenster, daß dessen Inhalt erneuert werden muß. Inhalt wird nicht automatisch erneuert!
`SMART_REFRESH`	Intuition meldet nur nach Größenveränderungen und erneuert sonst von alleine.
`NOCAREREFRESH`	Keine Meldungen, ob der Fensterinhalt erneuert werden muß.

`*FirstGadget`

Zeigt auf die Gadget-Struktur des ersten Gadgets einer Gadget-Liste.

`*CheckMark`

Zeigt auf eine Image-Struktur, in der eine »CheckMark« für das Menü dieses Fensters definiert ist. Steht hier eine Null, das heißt, ist kein Image definiert, so wird der »Haken« verwendet. (Falls jemand nicht weiß was eine CheckMark ist: Um ein Item im Menü zu kennzeichnen, sieht Intuition eine Funktion vor (siehe Menüs), die einfach zu handhaben ist. Will man nun ein Item kennzeichnen wird diese Checkmark von Intuition automatisch vor den ItemText eingefügt.)

`*Title`

Zeigt auf einen mit Null beendeten String und gibt den Titel des Fensters an, der in der Titelleiste der Umgrenzung erscheint. Steht hier eine Null, so wird kein Titeltext ausgegeben.

`*Screen`

Zeigt auf die Screen-Struktur des Screens in dem das Fenster erscheinen soll, falls nicht der WorkBench-Screen verwendet wird. Es ist wichtig, diese Adresse zu setzen, wenn bei Type `CUSTOMSCREEN` gesetzt ist, da sonst beim Öffnen des Windows ein Absturz droht. Screen ist aber erst nach dem Öffnen des Screens bekannt und muß dann in die NewWindow-Struktur übertragen werden, bevor das Fenster geöffnet wird.

`*BitMap`

Zeigt auf eine BitMap-Struktur (normalerweise Null).

`MinWidth,MinHeight,MaxWidth,MaxHeight`

Geben die minimale bzw. maximale Größe des Fensters, sofern dessen Größe verändert werden kann, das heißt das WINDOW-SIZING-Flag unter Flags gesetzt ist. Ist das Flag nicht gesetzt, so werden diese Werte nicht beachtet.

`Type`

Gibt an in welchem Screen das Fenster erscheinen soll, hier ist `WBENCHSCREEN` oder `CUSTOMSCREEN` zu setzen je nachdem, ob in dem WorkBench- oder in einem eigenen Screen gearbeitet werden soll.

Window-Funktionen

Zur Vereinfachung werden folgende Strukturen definiert:

```
struct NewWindow MyNewWindow = {...   };
struct Window *MyWindow;

MyWindow = OpenWindow(&MyNewWindow);
                              AØ
```

Diese Funktion entspricht in ihrem Aufruf der Screen-Funktion `OpenScreen()`. Sie öffnet ein in der MyNewWindow-Struktur definiertes Fenster und übergibt, falls es geöffnet werden konnte, einen Zeiger auf die Window-Struktur. Andernfalls wird eine Null übergeben. Das Öffnen eines Windows kann also mißlingen, auch wenn dies eher die Seltenheit bleibt. Meist ist es dann entweder auf einen Definitionsfehler oder auf mangelnden Speicherplatz zurückzuführen. An dieser Stelle sei noch bemerkt, daß, bevor an das Öffnen eines Screens oder Windows gedacht werden kann, zuerst das Intuition-Library geöffnet werden muß.

```
CloseWindow(MyWindow);
                AØ
```

schließt ein offenes Window wieder. Wie auch bei der Funktion `CloseScreen()` muß darauf geachtet werden, daß `MyWindow` tatsächlich auf eine Window-Struktur zeigt.

Fassen wir unsere bisherigen Erkenntnisse in einem kurzen Programm zusammen, das ein Window in einem eigenen Screen definiert (MyNewScreen- und MyNewWindow-Struktur werden nicht ausführlich definiert):

```
struct NewWindow MyNewWindow = {... ,CUSTOMSCREEN};
struct NewScreen MyNewScreen = {...              };
struct Window *MyWindow;
struct Screen *MyScreen;
struct IntuitionBase *IntuitionBase;

main()
{
 IntuitionBase=OpenLibrary("intuition.library",Ø);
 if(IntuitionBase==Ø)
    exit(FALSE);
 MyScreen=OpenScreen(&MyNewScreen);
 if(MyScreen==Ø)
 {
    CloseLibrary(IntuitionBase);
    exit(FALSE);
 }
 MyNewWindow.Screen=MyScreen;
 MyWindow=OpenWindow(&MyNewWindow);
 if(MyWindow==Ø)
 {
    CloseScreen(MyScreen);
    CloseLibrary(IntuitionBase);
    exit(FALSE);
    /* geschlossen werden sollte in der Reihenfolge, in der
    geöffnet wurde, um einen 'Guru' zu vermeiden. */
 }

 ...
```

```
/* Schließen des Fensters, des Screens und des Libs */
CloseWindow(MyWindow);
CloseScreen(MyScreen);
CloseLibrary(IntuitionBase);
}
```

```
WindowToBack (MyWindow);
WindowToFront(MyWindow);
                A0
```

Diese Funktionen entsprechen `ScreenToFront()` bzw. `ScreenToBack()`, legen also das Fenster vor oder hinter alle geöffneten Fenster. Fenster sind dem Screen untergeordnet in dem sie liegen. Sie können nachdem sie geöffnet wurden nicht mehr den Screen wechseln!

```
SizeWindow(MyWindow,dx,dy);
            A0     D0 D1
```

Vergrößert (Werte größer Null) bzw. verkleinert (Werte kleiner Null) das Fenster und `dx`- Pixel in `x`- und `dy`-Pixel in `y`-Richtung.

```
MoveWindow(MyWindow,dx,dy);
            A0     D0 D1
```

Bewegt das Fenster um `dx,dy`-Pixel über den Screen (in welche Richtung entscheidet wieder das Vorzeichen). Das Fenster kann natürlich nicht aus dem Screen herausbewegt werden!

```
SetWindowTitels(MyWindow,
                    A0
      WindowTitle,ScreenText);
          A1          A2
```

Macht es möglich, nach dem Öffnen eines Fensters dessen Titeltext zu verändern. Außerdem kann der Titelleiste des Screens ein Text zugeordnet werden, der immer erscheint, wenn das Fenster aktiviert wurde. `WindowTitel` und `ScreenText` können folgende Werte annehmen:

`0` – kein Name wird angezeigt,
`-1` – dieser Text wird nicht verändert,

ein mit Null beendeter Text-String, der übernommen wird.

Zum Schluß noch einige Bemerkungen zum Feststellen der Mausposition mit dem Intuition (siehe auch Programmroutine *frame.c*). In den noch zu besprechenden IDCMP_Flags wird dem Intuition mitgeteilt, welche Eingabeereignisse es über den Nachrichtenkanal an das Fenster melden soll. Mit Setzen des REPORTMOUSE-Flags meldet Intuition zum Beispiel bei Bewegungen der Maus die neuen Koordinaten. Dies kostet viel Zeit und ist deshalb ungünstig. Man kann die Mausposition auch einfacher feststellen. Man liest sie aus der Screen-Struktur, in der immer die aktuellen Werte stehen (`MyScreen→MouseX` bzw. `MyScreen→MouseY`) heraus. Dies Vorgehen hat allerdings den Nachteil, daß Werte verlorengehen können. Auf der anderen Seite ist es mit dem Aufruf

```
ReportMouse(MyWindow ,bool)
                A0      D0
```

möglich, die lästigen Meldungen der Maspositionen ab- bzw. anzustellen und so zu optimieren (`bool` entweder `TRUE` oder `FALSE`).

Beispielprogramm: *OpenWindow.c*

```
1 /*************************************************************************/
2 /* Oeffnen eines Windows                                                 */
3 /*                                                                       */
4 /* fuer Markt&Technik von H. Knappe                                      */
5 /*************************************************************************/
6
```

```
#include <exec/types.h>
#include <intuition/intuition.h>

struct IntuitionBase *IntuitionBase;

/* MyNewWindow ist eine NewWindow-Struktur          */
struct NewWindow MyNewWindow;
/* MyWindow ist ein Zeiger auf eine Window-Struktur */
struct Window *MyWindow;



main()
{
 LONG i;

 printf("Oeffnen eines Windows...\n");

 IntuitionBase=OpenLibrary("intuition.library",29);
 /* Library-Versionen ab 29 werden verwendet        */
 if(IntuitionBase==NULL)
 {
    printf("Intuition.library kann nicht geoeffnet werden!\n");
    exit(FALSE);
 }
 printf("Intuition.library offen -> ");

 /* MyNewWindow structure wird definiert. */
 MyNewWindow.LeftEdge= 50;     /* Fenster erscheint an Punkt (50,50)  */
 MyNewWindow.TopEdge= 50;
 MyNewWindow.Width= 400;       /* Fenster ist 400 Punkte breit        */
 MyNewWindow.Height= 100;      /* und 100 Punkte hoch                 */
 MyNewWindow.DetailPen= 0;     /* Fenster-Gadgets mit Farbregister 0  */
 MyNewWindow.BlockPen= 1;      /* Fenster-Begrenzung mit Register 1   */
 /* Schliessen des Fensters soll gemeldet werden.                     */
 MyNewWindow.IDCMPFlags= CLOSEWINDOW;
 /* Das Fenster soll geschlossen, vergroessert, verschoben und in die */
 /* 'Tiefe' gelegt werden koennen.  Das Fenster soll nach dem Oeffnen */
 /* aktiv sein  und im  SMART_REFRESH-Modus  soll der Inhalt erneuert */
 /* werden.                                                           */
 MyNewWindow.Flags= SMART_REFRESHACTIVATEWINDOWCLOSEWINDOWSIZING
                    WINDOWDRAGWINDOWDEPTH;
 MyNewWindow.FirstGadget= NULL; /* keine Gadgets                      */
 MyNewWindow.CheckMark= NULL;   /* keine eigene CheckMark             */
 MyNewWindow.Title= "Mein erstes Window"; /* Titel des Fensters     */
 MyNewWindow.Screen= NULL;      /* kein eigener Screen -> WBENCHSCREEN */
                                /* muß gesetzt sein!                  */
 MyNewWindow.BitMap= NULL;      /* keine eigene BitMap                */
 MyNewWindow.MinWidth= 50;      /* maximale und minimale Groesse des  */
 MyNewWindow.MinHeight= 50;     /* Fensters                           */
 MyNewWindow.MaxWidth= 320;
 MyNewWindow.MaxHeight= 200;
 MyNewWindow.Type= WBENCHSCREEN; /* Fenster erscheint im WBenchScreen */

 /* Oeffnen des  Fensters. Wenn das Ergebnis gleich Null ist kann das */
 /* Window nicht geoeffnet werden und das Programm sollte abbrechen.  */
 if((MyWindow=OpenWindow(&MyNewWindow))==NULL)
 {
    printf("Window kann nicht geoeffnet werden!\n");
    CloseLibrary(IntuitionBase);
    exit(FALSE);
 }
 printf("Fenster offen\n");

 /* Warten bis das Window geschlossen wird */
 while(GetMsg(MyWindow->UserPort)==0);

 /* Fenster schliessen                     */
 CloseWindow(MyWindow);
 printf("Fenster geschlossen.\n");
}
```

IDCMP-Flags

Wie oben schon angesprochen, sind es diese Flags, die bestimmen, welche Eingabeereignisse gemeldet werden sollen. Ein Eingabeereignis kann das Drücken einer Taste, das Anklicken des CloseGadgets oder ähnliches sein. Die Flags werden sowohl zur Angabe, als auch zur Auswertung benutzt! Da es sehr

viele IDCMP-Flags gibt, möchte ich mich auf die in den Programmen benutzten beschränken.

MOUSEBUTTONS

Ein Mausknopf wurde gedrückt, bzw. dieses soll gemeldet werden. In »Code« der Intui-Message-Struktur steht ein Wert, der angibt, ob und wie eine Maustaste gedrückt worden ist.

Es bedeuten:

SELECTDOWN	Rechte Maustaste wurde gedrückt.
SELECTUP	Rechte Maustaste wurde losgelassen.
MENUDOWN	Linke Maustaste wurde gedrückt.
MENUUP	Linke Maustaste wurde losgelassen.

GADGETUP / GADGETDOWN

Ein Gadget wurde angeklickt bzw. losgelassen. Die Kennung des Gadgets, um welches es sich gehandelt hat, steht in »Code«.

MENUPICK

Ein Menüpunkt wurde angewählt. In »Code« wird angegeben, um welches Menü es sich handelt. Steht hier MENUNULL, so wurde kein Menü gewählt.

NEWSIZE

Die Größe des Fenters wurde verändert.

RAWKEY

Eine Taste wurde gedrückt oder logelassen. In »Code« steht der Tastencode.

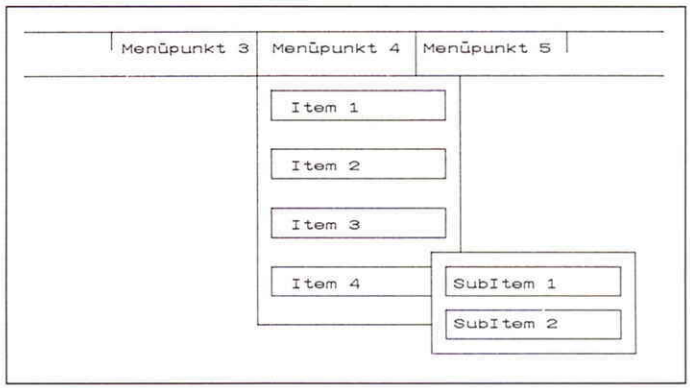
CLOSEWINDOW

Das CloseGadget wurde angeklickt.

8.1.3 Menüs

Zum Abschluß des Ausflugs ins Intuition stehen noch die Menüs. Sie bilden die Möglichkeit, sehr einfach Auswahlmenüs aufzubauen, und sind für ein unfangreicheres Programm unbedingt zu verwenden. Die Erklärung der anderen Intuition-Elemente wie Gadgets, Requester und Alerts würden hier zu weit führen.

Ich gehe davon aus, daß jeder, der dies liest, schon einmal mit dem Amiga gearbeitet hat und weiß, wie ein Menü aussieht, daß es sich in einzelne Menüpunkte unterteilt, die dann wiederum in Items und SubItems unterteilt sind (siehe Abbildung 8-2). Ein Menü ist immer einem Fenster zugeordnet, also ohne dies nicht möglich.



8-2 Aufbau eines Menüs

Es besteht die Möglichkeit, gleichzeitig bis zu 31 Menüpunkte darzustellen, von denen sich jedes in 63 Items unterteilen kann, die dann wieder jedes für sich in 31 SubItems unterteilt sein können. Ingesamt sind also bis zu 1953 verschiedene Anwahlpunkte gleichzeitig möglich.

Jedes Element erhält eine Kennung in Form eines 16-Bit-Wertes, der folgenden Aufbau hat:

```
Bit   1 2 3 4 5   6 7 8 9 10 11   12 13 14 15 16
      Menü        Item            SubItem
```

Steht hier also zum Beispiel 01000 100000 01100, so beschreibt dieser Wert das 6. SubItem des 1. Items des 2. Menüpunktes. Um zum Beispiel festzustellen, welcher Menüpunkt angewählt wurde, ist die Analyse eines solchen 16-Bit-Wertes nötig. Soll nur ein Menü durch die Kennung bestimmt werden, so müssen alle Item- und SubItem-Bits gesetzt sein. Dies liegt daran, daß die Zählung mit Null beginnt. Ist zum Beispiel gar nicht angewählt worden, so ist der 16-Bit-Wert gleich MENUNULL, das bedeutet, daß alle Bits gesetzt sind.

Die Initialisierung der Menüpunkte erfolgt mit Menu-Strukturen, die der Items und SubItems mit MenuItem-Strukturen. Wie die eigentliche Defintition aussieht, wird wohl am besten an dem Beispielprogramm zur Initialisierung und Abfrage von Menüs deutlich.

```
struct Menu
{
struct Menu *NextMenu;
SHORT  LeftEdge,TopEdge,Width,Height;
USHORT Flags;
BYTE *MenuName;
struct MenuItem *FirstItem;
}
```

*NextMenu

Zeigt auf die nachfolgende Menu-Struktur. Ist dies das letzte Menü, so steht hier Null.

LeftEdge, TopEdge, Width, Height

Definieren die Größe des Menüfeldes. TopEdge und Height werden nicht verwendet, das heißt, ein Menu hat immer die Höhe der Titlebar und beginnt an dessen Oberkante.

Flags

Steht hier MENUENABLED, so ist das Menü aktiv.

*MenuName

Zeigt auf den Text des Menüs.

*FirstItem

Zeigt auf die MenuItem-Struktur des ersten Items in diesem Menü.

```
struct  MenuItem
{
struct MenuItem *NextItem;
SHORT  LeftEdge,TopEdge,Width,Height;
USHORT Flags;
LONG   MutualExclude;
APTR   ItemFill;
APTR   SelectFill;
BYTE   Command;
struct MenuItem *SubItem;
USHORT NextSelect;
}
```

*NextItem

Ist ein Zeiger auf die MenuItem-Struktur des nächsten Items. Ist dies das letzte Item, steht hier Null.

LeftEdge,TopEdge,Width,Height

Geben die Lage und Größe des Items an.

Flags

COMMSEQ

Das Item soll über die Tastatur anwählbar sein (siehe Command).

ITEMTEXT

Das Item soll aus Text bestehen. Bei ItemFill muß dann ein Zeiger auf eine IntuiText-Struktur stehen.

ITEMENABLED

Bedeutet, daß dieses Item anwählbar ist.

CHECKIT/CHECKED

Soll ein Item oder SubItem mit einer »Check-Mark« versehen werden, so muß CHECKIT gesetzt sein. Intuition setzt dann, sobald das Item oder SubItem angewählt wird, den Flag CHECKED und versieht das Item mit einer CheckMark. Um diese wieder zu entfernen, muß CHECKED aus der entsprechenden MenuItem-Struktur entfernt werden.

MutualExclude

schließt zugleich bis zu 32 andere Items oder Sub-Items aus. Hier steht ein 16-Bit-Wert (WORD), dessen gesetzte Bits bis zu 16 Menü-, Item- bzw. Sub-Item-Punkte ausschließen, das heißt, sofern die Checkmarks gesetzt sind, werden diese gelöscht.

ItemFill

Zeigt entweder auf eine IntuiText-Struktur, falls ein Text angezeigt werden soll (bei »Flags« muß ITEMTEXT gesetzt sein!), oder auf eine Image-Struktur, die eine eigene Grafik definiert.

SelectFill

Analog ItemFill. Hier wird das Aussehen des Gadgets beim Anwählen bestimmt.

Command

Ist das COMMSEQ Flag bei »Flags« gesetzt, wird dieser Wert berücksichtigt. Hier steht ein Tastenzeichen im ASCII-Code, das ein Anwählen eines Menüs von der Tastatur aus möglich macht. Steht hier zum Beispiel 'B', so kann dieses Item durch Drücken der Tasten »B« und der rechten *»Amiga«*-Taste angewählt werden. Intuition blendet den Tastencode automatisch in das Item ein.

*SubItem

Zeigt auf eine MenuItem-Struktur, die das erste SubItem dieses Items beschreibt. Hat dies Item keine SubItems steht hier Null. Wichtig zu verstehen ist, daß sowohl Items als auch SubItems mit gleichen Strukturen definiert werden!

NextSelect

Enthält Zusatzinformationen des Intuition.

Der Text der Items und Subitems wird in IntuiText-Strukturen definiert. Diese Struktur hat folgenden Aufbau:

```
struct IntuiText
{
UBYTE  FrontPen,BackPen;
UBYTE  DrawMode;
SHORT  LeftEdge;
SHORT  TopEdge;
struct TextAttr *ITextFont;
UBYTE  *IText;
struct IntuiText *NextText;
}
```

FrontPen

Schriftfarbe mit der der Text gezeichnet wird.

BackPen

Farbe bestimmt das Farbregister des Schrift-hintergrundes bei JAM2.

DrawMode

Für Text gibt es drei verschiedene Darstellungsarten:

JAM1	Hintergrund bleibt unverändert.
JAM2	Schrift wird hinterlegt.
COMPLEMENT	Text wird komplementär zum Hintergrund eingeblendet.

`LeftEdge,TopEdge`

Gibt die Koordinaten des Textes an. Auf welchen Punkt sich diese Koordinaten beziehen, hängt von dem Objekt ab in dem die Struktur verwendet wird. Bei Menüs zum Beispiel immer auf die linke obere Ecke des Menüs, dessen Koordinaten bezüglich dem Screen in der Menu-Struktur definiert worden sind.

`*ITextFont`

Zeigt auf eine TextAttr-Struktur und bietet die Möglichkeit, für den Text eine eigene Schriftart zu wählen.

`*IText`

Ist ein Zeiger auf den eigentlichen Text.

`*NextText`

Zeigt auf eine weitere IntuiText-Struktur. Falls dies der letzte oder einzige Text ist, so steht hier eine Null.

Menü-Funktionen

Nachdem wir die Menüs und Items initialisiert haben, müssen wir sie noch einrichten. Dies geschieht mit der Funktion

```
SetMenuStrip(MyWindow,FirstMenu);
                A0        A1
```

(FirstMenu zeigt auf die erste Menu-Struktur der Liste.)

Wie wir jetzt erkennen, ist ein Menü also immer an ein bestimmtes Fenster gebunden. Bevor also Menüs existieren können, muß zuerst ein Fenster geöffnet werden.

Wollen wir ein Menü wieder entfernen, so geschieht dies mit

```
ClearMenuStrip(MyWindow);
                  A0
```

Um aus dem 16-Bit-Wert der Menü-Kennung das Menu, Item oder SubItem herausfinden zu können, wurden zur Vereinfachung folgende Makros definiert:

```
MyMenu=MENUNUM(Menükennung);     (0-31)
MyItem=ITEMNUM(Menükennung);     (0-63)
MySubitem=SUBNUM(Menükennung);   (0-31)
```

(Anmerkung: Die Zählung der Menüs, Items und Subitems beginnt immer mit Null.)

Beispielprogramm: *menu.c*

```
/*************************************************************************/
/* Beispiel fuer die Initialisierung, Installation und Abfrage von Menus */
/*                                                                       */
/* fuer Markt&Technik von H. Knappe                                      */
/*************************************************************************/

#include <exec/types.h>
#include <intuition/intuition.h>

#define ON ITEMTEXTITEMENABLEDHIGHCOMPCHECKITCHECKEDCOMMSEQ
#define OFF ITEMTEXTITEMENABLEDHIGHCOMPCOMMSEQ
#define OFFN ITEMTEXTITEMENABLEDHIGHCOMP

/* Definitionen der Menu-, Item-, SubItemnamen */
struct IntuiText Ende_Text={3,1,COMPLEMENT,1,1,0,"Ende",0};
struct IntuiText F1_Text={3,1,COMPLEMENT,3,1,0,"  320x200",0};
struct IntuiText F2_Text={3,1,COMPLEMENT,3,1,0,"  640x200",0};
struct IntuiText F3_Text={3,1,COMPLEMENT,3,1,0,"  320x400",0};
 struct IntuiText Funk1_Text={3,1,COMPLEMENT,1,1,0,"z^3-1=0",0};
 struct IntuiText Funk2_Text={3,1,COMPLEMENT,1,1,0,"z^2-c",0};
struct IntuiText Funk_Text={3,1,COMPLEMENT,1,1,0,"Funktion",0};

/* 2. Menu */
struct MenuItem MenuItem4=
 {0,0,33,113,10,OFF,0,&F3_Text,0,'3',0,0};
struct MenuItem MenuItem3=
 {&MenuItem4,0,22,113,10,OFF,0,&F2_Text,0,'2',0,0};
struct MenuItem MenuItem2=
 {&MenuItem3,0,11,113,10,OFF,0,&F1_Text,0,'1',0,0};
```

```
30   struct MenuItem SubItem2=
31    {0,98,1,90,10,OFFN,0,&Funk2_Text,0,0,0,0};
32   struct MenuItem SubItem1=
33    {&SubItem2,98,-8,90,10,OFFN,0,&Funk1_Text,0,0,0,0};
34  struct MenuItem MenuItem1=
35   {&MenuItem2,0,0,113,10,OFFN,0,&Funk_Text,0,0,&SubItem1,0};
36
37  /* 1. Menu */
38  struct MenuItem MenuItem5=
39   {0,0,0,75,10,OFFN,0,&Ende_Text,0,0,0,0};
40
41  struct Menu Menu2=
42   {0,80,0,75,0,MENUENABLED,"Format",&MenuItem1};
43  struct Menu Menu1=
44   {&Menu2,0,0,75,0,MENUENABLED,"Projekt",&MenuItem5};
45
46  struct NewWindow MyNewWindow=
47   {59,42,324,76,0,1,MENUPICK+CLOSEWINDOW,WINDOWDRAG+WINDOWDEPTH+WINDOWCLOSE+
48   ACTIVATE+NOCAREREFRESH,0,0,"Test-Window",0,0,50,50,640,200,WBENCHSCREEN};
49
50  struct IntuitionBase *IntuitionBase;
51  struct GfxBase *GfxBase;
52  struct IntuiMessage *MyMessage;
53  struct Window *MyWindow;
54
55
56
57  main()
58  {
59   ULONG class;
60   USHORT code;
61   int bool=0,menu,item,subitem;
62
63   /* Oeffnen der Libraries und des Windows */
64   if((GfxBase=OpenLibrary("graphics.library",29))==0)
65      exit(FALSE);
66   if((IntuitionBase=OpenLibrary("intuition.library",29))==0)
67   {
68      CloseLibrary(GfxBase);
69      exit(FALSE);
70   }
71   if((MyWindow=OpenWindow(&MyNewWindow))==0)
72   {
73      CloseLibrary(IntuitionBase);
74      CloseLibrary(GfxBase);
75      exit(FALSE);
76   }
77
78   SetMenuStrip(MyWindow,&Menu1); /* MenuLeiste wird installiert */
79   Move(MyWindow->RPort,15,20);
80   Text(MyWindow->RPort,"Window mit Menues",15);
81
82   for(;;)
83   {
84      /* Warten, bis etwas geschieht */
85      while((MyMessage=GetMsg(MyWindow->UserPort))==0);
86      class=MyMessage->Class;
87      code=MyMessage->Code;
88      switch(class)
89      {
90         /* was ist geschehen? */
91         case CLOSEWINDOW: CloseWindow(MyWindow);
92                           CloseLibrary(IntuitionBase);
93                           CloseLibrary(GfxBase);
94                           exit(TRUE);
95         case MENUPICK:    menu=MENUNUM(code);
96                           item=ITEMNUM(code);
97                           subitem=SUBNUM(code);
98                           switch(menu)
99                           {
100                             case 0: CloseWindow(MyWindow);
101                                     CloseLibrary(IntuitionBase);
102                                     CloseLibrary(GfxBase);
103                                     exit(TRUE);
104                                     break;
105                             case 1: switch(item)
106                             {
107                                case 0: switch(subitem)
108                                {
109                                   case 0: printf("Funktion z^3-1=0\n");
110                                           break;
```

```
111                                    case 1: printf("Funk-
tion 2^2-c\n");
112                                            break;
113                                    }
114                                 break;
115                                 case 1: printf("Format 1\n");
116                                        break;
117                                 case 2: printf("Format 2: %d\n",bool);
118                                        bool-=1;
119                                        bool=-bool;
120                                        if(bool==1)
121                                           MenuItem3.Flags=ON;
122                                        if(bool==0)
123                                           MenuItem3.Flags=OFF;
124                                        break;
125                                 case 3: printf("Format 3\n");
126                                        break;
127                              }
128                              break;
129                           }
130     }
131  }
132 }
```

8.2 Die Grafikfunktionen

Um diese Funktionen von C aus benützen zu können, muß man das Grafik-Library öffnen. Dies entspricht dem Vorgehen beim Öffnen des Intuition-Library, bevor Routinen des Intuition wie zum Beispiel `OpenScreen()` aufgerufen werden können. Ist das Grafik-Library nicht geöffnet, so kann sonst ein sicheres »Finish all Disk Activity« erwartet werden.

```
struct GfxBase *GfxBase;
GfxBase=OpenLibrary
("graphics.library",0);
if(GfxBase==0) exit(FALSE);
```

Für die meisten Grafik-Funktionen ist die sogenannte RastPort-Struktur von großer Bedeutung. Einen Zeiger auf diese erhält man aus der Screen- oder Window-Struktur. Die Koordinaten, die angegeben werden müssen, beziehen sich immer auf den »Ursprung« des in der RastPort-Struktur definierten Gebietes, also zum Beispiel auf die linke obere Ecke eines Screens oder Fensters.

```
struct RastPort *MyRastPort;
für einen Screen:
MyRastPort=MyScreen->RastPort;
für ein Window:
MyRastPort=MyWindow->RPort;
```

```
SetRGB4(MyViewPort, n , r , g , b)
          A0       D0  D1  D2  D3
```

Bisher ist uns zwar bekannt, wie ein Screen geöffnet wird und daß dessen Tiefe die Anzahl der möglichen Farben bestimmt. Wie werden aber die Farben eigentlich gesetzt? Dies geschieht mit dieser Funktion. `n` gibt das Farbregister an, dessen Farbe definiert werden soll (wenn zum Beispiel 32 Farben möglich sind, kann `n` Werte von 0 bis 31 annehmen). `r`, `g` und `b` geben die Rot-, Grün- und Blauanteile der Farbe an (hier sind Werte von 0 bis 15 für `r`, `g` bzw. `b` möglich- (`r,g,b`) (`15,15,15`) ist also zum Beispiel Weiß, (`0,0,0`) Schwarz und (`15,0,0`) Rot).

Eines der wichtigsten Elemente für den Bildaufbau ist die ViewPort-Struktur, die jeder Screen besitzt. Man erhält sie also aus der Screen-Struktur.

```
MyViewPort=&(MyScreen->ViewPort);
```

Das nächste Problem taucht nun aber auf, wenn man mit einer bestimmten Farbe zum Beispiel einen Punkt setzen will. Wie wird die Farbe gewählt?

```
SetAPen(MyRastPort,pen);
          A1       D0
```

Wählt für die folgenden Grafikfunktionen den »Stift« (Werte von 0–63 sinnvoll, falls der EXTRA-HALFBRITE-Modus verwendet wird), mit dem gemalt werden soll. Die Nummer des Stiftes stimmt mit dem Farbregister überein, solange kein HAM-Modus angeschaltet ist.

Es gibt die Möglichkeit, gleichzeitig mit zwei verschiedenen Stiften zu malen. Zum Wählen des zweiten Stiftes wird die Funktion

```
SetBPen(MyRastPort,pen);
          A1      DØ
```

benutzt.

Bei der fraktalen Grafik stellt sich oft das leidige Problem, wie man die notwendigen Parameter übergibt. Im speziellen ist es schwierig, die Koordinaten für einen Ausschnitt aus einem fertigen Bild herauszufinden und einzugeben. Es liegt nahe, diese Aufgabe dadurch zu vereinfachen, daß wir in dem berechneten Bild einen interessanten Ausschnitt »framen« (einrahmen) können, dessen Koordinaten dann automatisch umgerechnet werden. (Ein Programm–*frame.c*, das die Funktionen übernimmt, ist im Programmteil erklärt.)

Bei dem Einrahmen taucht nun aber das Problem auf, eine Linie zu ziehen, die möglichst den Hintergrund nicht verändert. Der Amiga bietet dazu einen Zeichen-Modus an, der beim Zeichnen die Punkte nicht übermalt, sondern nur invertiert. Nach einem zweimaligen Aufruf der gleichen Operation ist also das alte Bild wieder ohne Veränderung erreicht.

Es sind folgende Zeichenmodi möglich (sind zugleich Flags):

`JAM1` Dies ist die normale Malart. Alle Punkte werden mit der Farbe, des »APen« gemalt.

`JAM2` Wie bereits angesprochen, kann gleichzeitig mit zwei Stiften gemalt werden, dazu ist dieser Modus zu setzen.

`COMPLEMENT` Diesen Flag müssen wir für das »Framen« setzen. Jetzt wird beim Malen der Bildschirminhalt invertiert.

`INVERSEVID` Dies ist interessant für die Ausgabe von Texten und bestimmt, wie der Text erscheinen soll.

Das Setzen der Flags erfolgt mit Aufruf von

```
SetDrMd(MyRastPort,MyMode);
          A1        DØ
```

Der gesetzte Zeichen-Modus bleibt für einen RastPort solange erhalten, bis ein neuer mit dieser Funktion gewählt wird. `MyMode` sind die oben angeführten Flags. An dieser Stelle noch eine Anmerkung: Alle bisher verwendeten Flags wurden in den Include-Files mit `#define..` vordefiniert und können dort aufgefunden werden. Außerdem ist dann selbstverständlich, daß bei ihrer Verwendung die entsprechenden Files mit `#include<>` eingebunden werden müssen. Die MyMode-Flags findet man zum Beispiel in *rastport.h*. Wichtig ist vor allem, daß man die Schreibweise genau einhält, auch die Groß- und Kleinschreibung, da der Compiler diese sonst nicht erkennt.

`SetDrMd(MyRastPort,COMPLEMENT);`
setzt zum Beispiel den invertierenden Modus. Er bleibt solange erhalten, bis er mit `SetDrMode(MyRastPort,JAM1);` abgeschaltet wird.

Um einen Text in das durch MyRastPort bestimmte Gebiet (also zum Beispiel einen Screen) zu schreiben, gibt es die Funktion

```
Text(MyRastPort,Text,Länge);
       A1       AØ    DØ
```

Dabei ist Text ein Zeiger auf einen ASCII-TextString und Länge die Anzahl der Zeichen, die er enthält. Die Ausgabe des Textes erfolgt mit der diesem RastPort zugeordneten Schriftsatzes.

Was tun, wenn die Länge eines Textes in Pixeln bestimmt werden soll? Nichts leichter als das! Ein kurzer Aufruf von

```
PixelLänge=TextLength
                (MyRastPort,Text,Länge);
                   A1       AØ    DØ
```

löst das Problem. Text ist wieder ein Zeiger auf einen ASCII-String, Länge die Anzahl dessen Zeichen und PixelLänge dessen wirkliche Länge in Punkten, die er nach einer Ausgabe einnimmt.

Diese beiden Funktionen gehören zu den Grafikfunktionen; es gibt aber auch noch einen Intuition-Aufruf zur Texteingabe, der hier noch zur Vollständigkeit erwähnt werden soll:

```
PrintIText
      (MyRastPort,MyIntuiText, x , y );
          AØ          A1       DØ  D1
```

gibt einen Text aus. MyIntuiText ist dabei ein Zeiger auf eine IntuitionText-Struktur, x und y die Koordinaten des Textes relativ zu (0,0).

Bei dieser Intuition-Funktion werden wir daran erinnert, daß der auszugebende Text in dem Ausgabegebiet positioniert werden muß. Bei der PrintIText()-Funktion können wir diese Koordinaten direkt angeben, wie aber bei dem Aufruf Text()? Wie schon weiter oben angesprochen, ist die Vorstellung mit einem Stift zu malen durchaus sinnvoll. Bevor wir eine Linie ziehen können, muß der Stift erst auf eine bestimmte Stelle positioniert werden. Diese Positionierung erfolgt mit dem Aufruf

```
Move(MyRastPort, x , y );
       AØ        DØ  D1
```

x und y sind dabei die Pixelkoordinaten des Punktes, auf den der Stift gesetzt werden soll. (Bis jetzt haben wir noch nichts gemalt!) Wollen wir also zum Beispiel einen Text an eine bestimmte Stelle schreiben, so müssen wir zuerst den Stift mit Move() positionieren und dann den Text zum Beispiel mit Text() ausgeben.

```
WritePixel(MyRastPort, x , y );
             A1       DØ  D1
```

Setzt einen Punkt mit der Farbe des »APen« an x,y. Hier muß also nicht erst mit Move() der »Zeichen-Cursor« auf eine bestimmt Stelle gesetzt werden.

```
Draw(MyRastPort, x , y );
       AØ       DØ  D1
```

Macht es möglich, eine Linie von der augenblicklichen Stiftposition zum Punkt (x,y) zu ziehen. Welche Farbe die Linie erhält, wird durch den Zeichenmodus mit SetDrMd() vorbestimmt. Bei JAM1 wird sie mit der Farbe des »APen« gezeichnet.

Will man Polygone zeichnen, so wird es nötig, mehrere Male hintereinander die Routine Draw() aufzurufen. Zur Vereinfachung bietet das Betriebssystem des Amiga folgende Funktion:

```
PolyDraw(MyRastPort,Anzahl,Daten);
           A1         DØ      AØ
```

Hier werden `Anzahl` Linien, deren Eckpunkte in `Daten` definiert wurden, mit einem Aufruf gezogen.

`Anzahl` ist also die Anzahl der Linien die gezogen werden sollen. Daten zeigt auf ein Datenfeld in dem die Eckpunkte des Polygons als (x,y)-Paare hintereinander abgelegt sind. Will man zum Beispiel ein Rechteck zeichnen:

```
short daten[]={Ø,Ø,1Ø,Ø,1Ø,1Ø,Ø,1Ø};
short anz=4;
...
Move(MyRastPort,Ø,1Ø);
SetAPen(MyRastPort,1);
PolyDraw(MyRastPort,anz,&daten[Ø]);
```

Mit den bisherigen Funktionen ist es uns nun möglich, Punkte zu setzen und Linien zu ziehen. Zum Malen von gefüllten Flächen sind die folgenden Routinen gedacht:

```
SetRast(MyRastPort,Farbreg);
            A1          DØ
```

füllt das durch `MyRastPort` definierte Gebiet mit der Farbe aus dem Register `Farbreg` vollständig aus. Diese Funktion macht es zum Beispiel sehr einfach, ein ganzes Bild zu löschen.

```
Flood(MyRastPort,Modus, x , y );
          A1        D2    DØ  D1
```

Füllt Flächen auf zwei verschiedene Weisen.

`Modus 0:`
Das zu füllende Gebiet ist von einer Linie einer bestimmten Farbe begrenzt. Um nun dieses Gebiet zu füllen ist mit `SetOPen()` die Farbe der Begrenzung zu bestimmen und ein Punkt (x,y) im Gebiet zu übergeben. Die Funktion füllt dann selbstständig die Fläche.

`Modus 1:`
Nun ist die zu füllende Fläche bereits einfarbig gefüllt und soll nun mit einer anderen Farbe oder einem Muster übermalt werden. Wieder wird mit (x,y) ein Punkt innerhalb der Fläche angegeben.

Die Füllfarbe ist bei beiden Arten mit `SetAPen` zu setzen; ein Muster mit `SetAfPt()`.

Um Rechtecke zu füllen ist auch eine einfache Routine vorhanden, die keine Initialisierung eines Zwischenspeichers benötigt:

```
RectFill
      (MyRastPort,xmin,ymin,xmax,ymax);
           A1      DØ   D1   D2   D3
```

füllt das durch die beiden Punkte (`xmin,ymin`) und (`xmax,ymax`) beschriebene Rechteck mit einer Farbe (`SetAPen()`) oder einem Muster (`SetAfPt()`). Mit `SetOPen()` kann außerdem noch die Farbe der Begrenzungslinie des Rechtecks gewählt werden.

Diese Füllfunktionen kamen alle ohne einen Zwischenspeicher aus, der von uns selbst definiert werden muß. Die *Areafill*-Funktionen benötigen dagegen einen solchen. Ihre Initialisierung ist deshalb ein wenig aufwendiger.

Zu allererst müssen wir Speicherplatz für eine BitPlane reservieren, in der die Funktionen dann arbeiten können. Dies geschieht durch Aufruf von

```
Speicher=AllocRaster( x, y );
  DØ                  DØ D1
```

Speicher ist ein Zeiger auf das erste Byte des angeforderten Speichers, der die Größe (x, y) hat. Sollte es aus Speicherplatzgründen dem Betriebssystem nicht möglich sein, den notwendigen Speicher zu reservieren, so kehrt die Routine mit Null zurück. In diesem Fall

können dann natürlich auch die nachfolgenden Funktionen nicht verwendet werden.

Mit dem Makro `RASSIZE(x,y)` ist es möglich, den notwendigen Speicherplatz der mit AllocRaster angeforderten BitPlane zu bestimmen. Die Größe wird in der Funktion

```
Addresse=InitTmpRas
  DØ
                (MyTmpRas,Speicher,Größe);
                  AØ       A1       DØ
```

benötigt. Sie initialisiert die TmpRas-Struktur. `Speicher` zeigt auf den bereits reservierten Speicherplatz, `Größe` ist dies Anzahl der Bytes des reservierten Speichers, `Addresse` zeigt auf die neu initialisierte Struktur und `MyTmpRas` zeigt auf die zu initialisierende TmpRas-Struktur.

Nun muß noch die AreaInfo-Struktur initialisiert werden.

```
InitArea(MyAreaInfo,Buffer,Anzahl);
           AØ         A1       DØ
```

`MyAreaInfo` zeigt dabei auf die zu initialisierende Struktur, `Buffer` auf einen Speicherbereich für die Eckpunkte der Flächenbegrenzung (Achtung: Jeder Eckpunkt benötigt fünf Byte!) und `Anzahl` gibt die Zahl der maximal möglichen Eckpunkte an.

Zu guter letzt muß die AreaInfo-Struktur und die TmpRas-Struktur noch in der *RastPort*-Struktur vermerkt werden.

```
MyRastPort->TmpRas=Addresse;
MyRastPort->AreaInfo=&MyAreaInfo;
```

Diese ganzen Initialisierungen sind zwar recht aufwendig, sehen aber schwieriger aus als sie sind. Am besten versteht man das Vorgehen an Hand eines Studiums des entsprechendes Beispielprogramms.

Nachdem nun alle notwendigen Strukturen und Speicherplätze eingerichtet sind, kann's losgehen:

```
AreaMove(MyRastPort, x, y );
           A1       DØ D1
```

setzt den Startpunkt einer neuen Flächenbegrenzung. Mit

```
Fehler=AreaDraw(MyRastPort, x, y );
  DØ                 A1      DØ D1
```

wird ein Eckpunkt der Fläche angegeben. Sollte die Anzahl der zulässigen Eckpunkte erreicht sein, so kehrt die Funktion mit `-1` zurück, ansonsten mit Null.

Um nun die Fläche zu schließen und auszugeben ruft man zum Schluß

```
AreaEnd(MyRastPort);
          A1
```

auf. Das durch die Eckpunkte begrenzte Gebiet wird geschlossen und die gefüllte Fläche in den eigentlichen RastPort, also das Bild übertragen. Außerdem wird zugleich der Arbeitsspeicher gelöscht und eine neue Fläche kann erzeugt werden.

Beispielprogramm: *areaFill.c*

```
/*****************************************************************************/
/* Demo fuer AreaFill-Funktionen                                             */
/* fuer Markt&Technik von H. Knappe                                          */
/*****************************************************************************/

#include <exec/types.h>
#include <intuition/intuition.h>
#include <graphics/gfxmacros.h>

#define N 1Ø /* maximales Polygon mit N Ecken */

struct IntuitionBase *IntuitionBase;
struct GfxBase *GfxBase;
```

```
14 struct NewWindow MyNewWindow=
15  {1Ø,1Ø,2ØØ,1ØØ,Ø,1,NEWSIZE+CLOSEWINDOW,WINDOWDRAG+WINDOWDEPTH+
16  WINDOWCLOSE+WINDOWSIZING+ACTIVATE+NOCAREREFRESH+GIMMEZEROZERO,Ø,Ø,
17  "AreaFill Demo",Ø,Ø,5Ø,5Ø,64Ø,25Ø,WBENCHSCREEN};
18 struct Window *MyWindow;
19 struct RastPort *MyRastPort;
2Ø struct IntuiMessage *MyMessage;
21 struct AreaInfo MyAreaInfo;
22 struct TmpRas MyTmpRas;
23
24 UWORD buffer[5*N];
25 long  *speicher;
26
27
28 main()
29 {
3Ø  o_(); /* Oeffnen der Libraries und des Windows */
31  for(;;)
32  {
33     while((MyMessage=GetMsg(MyWindow->UserPort))==Ø)
34        area_();  /* Solange nichts passiert */
35     if(MyMessage->Class==CLOSEWINDOW)
36        break;
37     SetRast(MyRastPort,Ø); /* Fensterinhalt loeschen */
38  }
39  ReplyMsg(MyMessage);
4Ø  c_();
41 }
42
43
44 area_()
45 {
46  short xØ,yØ,x,y,x1,y1,col,n,i;
47
48  col=RangeRand(4);
49  SetAPen(MyRastPort,col);   /* Fuellfarbe */
5Ø  SetOPen(MyRastPort,3-col); /* Randfarbe */
51  /* Anpassung an die augenblickliche Fenstergroesse */
52  xØ=MyWindow->Width,yØ=MyWindow->Height;
53  n=RangeRand(N);
54  if(n<3) n=3;
55  x1=RangeRand(xØ),y1=RangeRand(yØ);
56  AreaMove(MyRastPort,x1,y1);
57  for(i=Ø;i<n-1;++i)
58  {
59     x=RangeRand(xØ),y=RangeRand(yØ);
6Ø     AreaDraw(MyRastPort,x,y);
61  }
62  AreaDraw(MyRastPort,x1,y1);
63  AreaEnd(MyRastPort);
64 }
65
66
67 o_()
68 {
69  if((IntuitionBase=OpenLibrary("intuition.library",Ø))==Ø)
7Ø     c_();
71  if((GfxBase=OpenLibrary("graphics.library",Ø))==Ø)
72     c_();
73  if((MyWindow=OpenWindow(&MyNewWindow))==NULL)
74     c_();
75  MyRastPort=MyWindow->RPort;
76  init_fill();
77 }
78
79 init_fill()
8Ø {
81  if((speicher=AllocRaster(64Ø,25Ø))==Ø)
82     c_();
83  MyRastPort->AreaInfo=&MyAreaInfo;
84  MyRastPort->TmpRas=InitTmpRas(&MyTmpRas,speicher,RASSIZE(64Ø,25Ø));
85  InitArea(&MyAreaInfo,&buffer[Ø],N);
86 }
87
88 c_()
89 {
9Ø  if(speicher!=Ø) FreeRaster(speicher,64Ø,25Ø);
91  if(MyWindow!=Ø) CloseWindow(MyWindow);
92  if(GfxBase!=Ø) CloseLibrary(GfxBase);
93  if(IntuitionBase!=Ø)  CloseLibrary(IntuitionBase);
94  exit(Ø);
95 }
```

Zum Schluß der Grafikroutinen noch zwei Funktionen, die ein direktes Arbeiten mit dem Blitter ermöglichen:

```
BltClear(Speicher,Anzahl,Flag);
           A1       DØ     D1
```

löscht die Anzahl Bytes eines durch Speicher bestimmten Bereichs. (Speicher zeigt auf das erste Byte dieses Gebietes.) Der Flag gibt an, ob solange, bis der Bereich gelöscht ist (1), gewartet oder gleich weitergearbeitet werden soll (0).

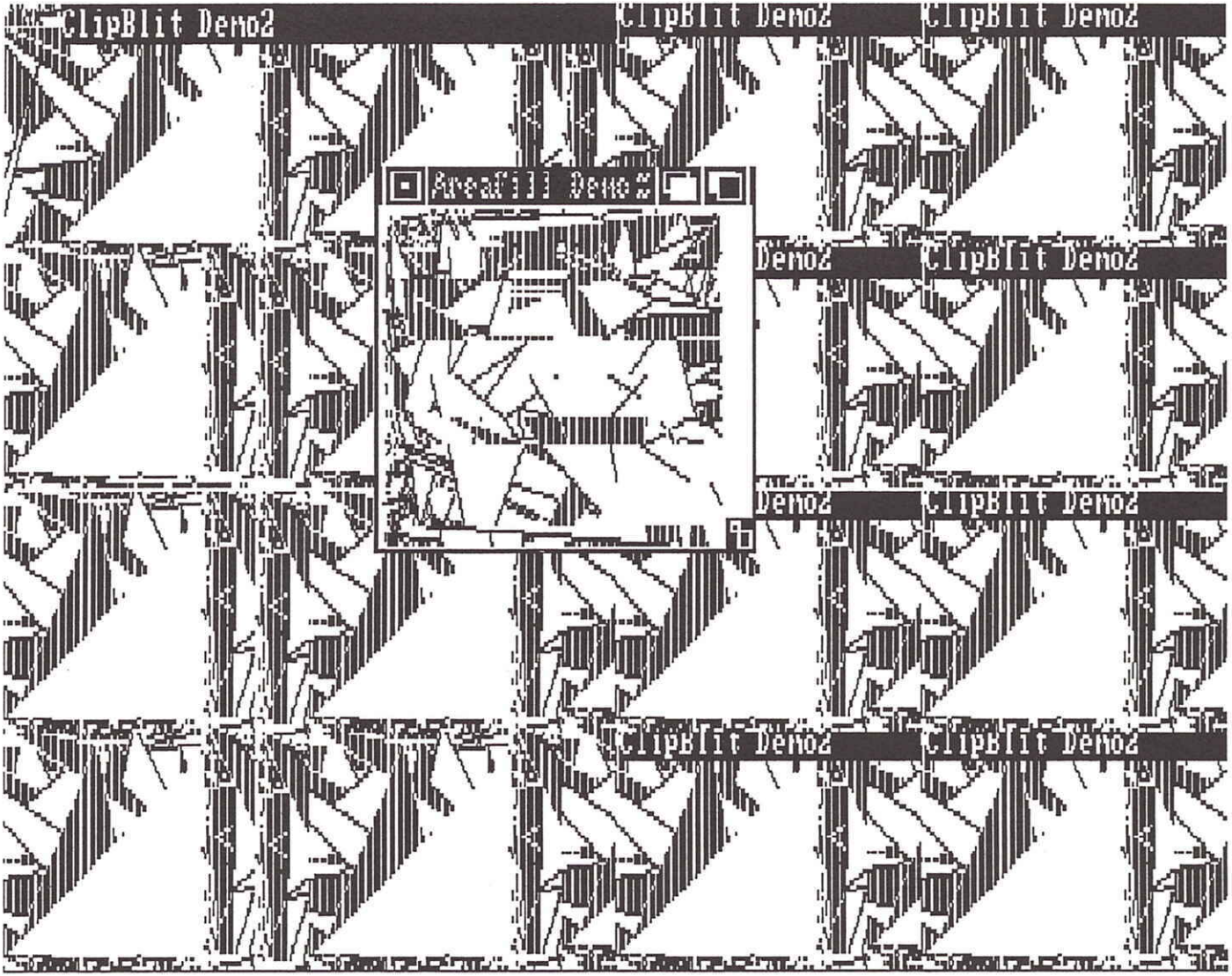
```
ClipBlit(QuellRPort,qx,qy,ZielRPort,
              AØ      DØ D1     A1
zx,    zy,    Breite,    Höhe,    Modus);
D2     D3      D4         D5       D6
```

Kopiert einen rechteckigen Bildausschnitt eines RastPort in einen anderen. `QuellRPort` zeigt auf den RastPort aus dem kopiert, `ZielRPort` auf den RastPort des Gebiets, in den der Ausschnitt geblittet werden soll. Die Größe des Ausschnitts wird mit `Höhe` und `Breite` angegeben. An welcher Stelle er liegen soll, bestimmt der Punkt (`qx,qy`) für die Quelle und (`zx,zy`) für das Ziel. Mit `Modus` kann angegeben werden, ob direkt (192) oder invertiert (48) kopiert werden soll.

Zu dieser Funktion gibt es ein Beispielprogramm, das den Umgang mit dieser verständlich macht. Es wird ein bestimmter Bereich um die Maus 16mal in ein Backdrop-Fenster geblittet. Erstaunlich ist vor allem die Geschwindigkeit des Blitters! Wie man sich vorstellen kann, gibt es natürlich Rückkopplungen, die vom Ausschnittsinhalt abhängen. Es kann zu einem Rauschen kommen, das ganz verschiedenes Aussehen aufweisen kann. Die dabei entstehenden Strukturen können durch Bewegungen der Maus stabilisiert und manipuliert werden. Dieser Effekt ist durchaus ähnlich dem Verhalten einer rekursiven Reihe und kann sehr einfach zeigen, was für seltsame Erscheinungen auftreten.

Beispielprogramm: *clip2.c*
(siehe Abbildung 8-3)

```
1  /*********************************************************************/
2  /* Demo2 fuer ClipBlit()                                             */
3  /* fuer Markt&Technik von H. Knappe                                  */
4  /*********************************************************************/
5
6  #include <exec/types.h>
7  #include <intuition/intuition.h>
8
9  #define WIDTH  64Ø
10 #define HEIGHT 25Ø
11 #define ANZ 4
12
13 struct IntuitionBase *IntuitionBase;
14 struct GfxBase *GfxBase;
15 struct NewWindow MyNewWindow=
16  {Ø,Ø,WIDTH,HEIGHT,Ø,1,CLOSEWINDOW,WINDOWCLOSE+ACTIVATE+NOCAREREFRESH+
17  BACKDROP,Ø,Ø,"ClipBlit Demo2",Ø,Ø,Ø,Ø,Ø,Ø,WBENCHSCREEN};
18 struct Window *MyWindow;
19 struct Screen *WbScreen;
20 struct RastPort *MyRastPort;
21 struct RastPort *WbRastPort;
22 struct IntuiMessage *MyMessage;
23
24 short mode=ØxCØ; /* nicht invertiert */
25
26
27 main(argc,argv)
28 int argc;
29 char *argv[];
30 {
31  if((argc==2)&&(*argv[1]=='i')) mode=Øx3Ø; /* Ausgabe invertiert? */
32  o_(); /* Oeffnen der Libraries und des Windows */
33  while((MyMessage=GetMsg(MyWindow->UserPort))==Ø)
34     clip_(); /* Solange nichts passiert */
```



8-3 Das Demo-Programm clipbit.c

```
35  ReplyMsg(MyMessage);
36  c_();
37 }
38
39
40 clip_()
41 {
42  short Mausx,Mausy,i,j;
43
44  Mausx=WbScreen->MouseX,Mausy=WbScreen->MouseY;
45  if(Mausx>(640-WIDTH/ANZ)) Mausx=640-WIDTH/ANZ;
46  if(Mausy>(250-HEIGHT/ANZ)) Mausy=250-HEIGHT/ANZ;
47  for(i=0;i<ANZ;++i)
48     for(j=0;j<ANZ;++j)
49        ClipBlit(WbRastPort,Mausx,Mausy,MyRastPort,i*WIDTH/ANZ,j*HEIGHT/ANZ,
50                 WIDTH/ANZ,HEIGHT/ANZ,mode);
51 }
52
53
54 o_()
55 {
56  if((IntuitionBase=OpenLibrary("intuition.library",0))==0)
57     c_();
58  if((GfxBase=OpenLibrary("graphics.library",0))==0)
59     c_();
60  if((MyWindow=OpenWindow(&MyNewWindow))==NULL)
61     c_();
62  MyRastPort=MyWindow->RPort;
63  WbScreen=MyWindow->WScreen;
64  WbRastPort=&(WbScreen->RastPort);
65  ShowTitle(WbScreen,FALSE);
66 }
67
68 c_()
69 {
70  if(MyWindow!=0) CloseWindow(MyWindow);
71  if(GfxBase!=0) CloseLibrary(GfxBase);
72  if(IntuitionBase!=0) CloseLibrary(IntuitionBase);
73  exit(0);
74 }
```

8.3 DOS

Zum Abschluß möchte ich noch kurz auf den Umgang mit der Disk eingehen. Hier sind nur die wichtigsten Funktionen aufgeführt, die auch in den Programmen Verwendung gefunden haben. Sie reichen sicher aus, mit Files auf der Disk umzugehen.

Als erstes einige allgemeine Bemerkungen: DOS steht für den englischen Ausdruck – Disk Operating System. Diese Bezeichnung scheint in erster Linie zu bedeuten, daß das DOS die Arbeit mit der Floppy verwaltet. Beim Amiga sind jedoch die Routinen des DOS für alle Aus- und Eingabegeräte wie zum Beispiel RAM-Disk, Parallele, Serielle Schnittstelle, Konsole etc. zuständig.

Daß diese Arbeit nicht unbedingt einfach ist, wird verständlich, wenn man die Multitaskingfähigkeit des DOS berücksichtigt, denn mehrere Programme können ja völlig unabhängig voneinander gleichzeitig auf verschiedene Geräte zugreifen. Besonders problematisch wird es, wenn zwei Programme gleichzeitig auf dasselbe Laufwerk zugreifen. Abwechselnd wird dann jedem Programm von der Diskette ein Block übergeben. Während man diese Arbeitsweise des Multitaskings natürlich sonst nicht bemerkt, schrappt der Kopf des Laufwerks über die Diskette und gibt gerade bei älteren Laufwerken einen ohrenbetäubenden Lärm von sich. Jetzt hilft meist nur noch ein Reset, um die Floppy zu schonen. Dies ist schon ärgerlich, man stelle sich nun aber einmal vor, zwei Programme könnten gleichzeitig in ein File schreiben – ein Daten-Chaos wäre zu erwarten. Deshalb verhindert das DOS auch diese Art des Zugriffs. Um zu kennzeichnen, ob ein File (=Datei) noch von anderen Programmen bearbeitet werden kann, verteilt das DOS

»Schlösser«, die das File »verschließen« können. Es gibt zwei Typen:

Das »gemeinsame Lese-Schloß« (Shared Read Lock) und das »ausschließende Schreib-Schloß« (Exclusive Write Lock). Bei der ersten Art können mehrere Anforderer gleichzeitig eine Datei lesen, während bei der zweiten nur ein Programm in sie schreiben kann. Angefordert wird ein solches Schloß mit der Funktion

```
MyLock = Lock(FileName,Art);
  DØ              D1     D2
```

`MyLock` zeigt auf eine Lock-Struktur, die uns aber im allgemeinen nicht interessiert. Sie ist nur für das DOS von Bedeutung. FileName zeigt auf einen String, der den Namen der Datei angibt. Art gibt schließlich an, um welche Art von Schloß es sich handelt. Hier ist `ACCESS_READ` für ein Lese-Schloß und `ACCESS_WRITE` für ein Schreib-Schloß vorgesehen. War es dem DOS nicht möglich, ein Schloß zuzuweisen, entweder weil schon ein Schreib-Schloß vergeben war, oder die Datei nicht existiert, ist `MyLock` gleich Null.

Um ein Schloß wieder freizugeben, wenn wir es nicht mehr brauchen, rufen wir auf:

```
Unlock(MyLock);
         D1
```

Es ist wichtig ein Schreib-Schloß wieder freizugeben (siehe auch `Close();`), da sonst das File als »Leiche« bestehen bleibt, das heißt, es kann entweder nicht mehr gelöscht werden oder versagt jeden Zugriff. Noch eine Anmerkung: Bevor die vordefinierten Flags verwendet werden können, muß wie üblich ein Include-File ins Programm eingebunden werden. Beim Dos stehen die notwendigen Definitionen in dem File *libraries/dos.h*!

Bisher hilft uns das Schloß noch nicht viel. Nun können wir uns aber vom DOS einige Informationen über die Datei geben lassen:

```
succ = Examine(MyLock,MyFileInfoBlock);
                 DØ      D1        D2
```

Bei diesem Aufruf schreibt das DOS Informationen über die Datei in den Speicherbereich MyFileInfoBlock. `succ` gibt an, ob diese Untersuchung erfolgreich (ungleich Null) war. `MyLock` zeigt auf eine Lock-Struktur, die uns das DOS zuvor mit der Funktion Lock übergeben hatte. `MyFileInfoBlock` zeigt auf eine FileInfoBlock-Struktur. Sie enthält die folgenden Elemente:

```
struct FileInfoBlock
{
LONG fib_DiskKey;
LONG fib_DirEntryType;  /* <Ø dann Datei, sonst Directory */
char fib_FileName[1Ø8];  /* Dateiname (max. 3Ø Zeichen) */
LONG fib_Protection;
LONG fib_EntryType;
LONG fib_Size;          /* File-Länge in Bytes */
LONG fib_NumBlocks;     /* Anzahl der belegten Blocks */
struct DateStamp fib_Date;
char fib_Comment[116];
}
```

Für uns interessant ist in erster Linie die Länge des Files, also `fib_Size`, da wir zum Beispiel ein gepacktes Bild-File ganz in den Speicher laden wollen, bevor wir es aufzubereiten beginnen (enormer Geschwindigkeitsgewinn! – Allgemein sollte man versuchen, nur größere Datenmengen vom DOS schreiben zu lassen, um die recht langsamen DOS-Operationen zu vermeiden!).

An dieser Stelle eine Warnung: Das Amiga-DOS ist aus dem Betriebssystem Tripos entstanden (siehe Artikel aus der Beilage zu

CHIP vom July 1987 – Vom Betriebssystem Tripos zum Amiga-DOS). Es arbeitet mit einer etwas anderen Daten-Struktur, so daß es beim Arbeiten mit dem DOS zu Schwierigkeiten kommen kann. Ich hatte zum Beispiel das folgende Problem: Ich wollte die Länge eines Files feststellen und ging folgendermaßen vor:

```
struct FileInfoBlock MyInfoBlock;
long   MyLock;
...

if((MyLock=Lock("test.prg",ACCESS_READ))==Ø)
   close_all(); /* Fehler -> Programm verlassen */
if(Examine(MyLock,&MyInfoBlock)==Ø)
   close_all();
printf("File-Länge: %d\n",MyInfoBlock.fib_Size);
...
```

Ganz klar, das müßte funktionieren. Wie zu erwarten tat das Programm auch das, was es sollte und druckte brav die Länge der Datei *test.prg* aus, sofern diese vorhanden war. Also baute ich dies kurze Programm in ein größeres ein. Zuerst ging auch alles gut, bis ich, nachdem ich einige Änderungen am Programm vorgenommen hatte, festellen mußte, daß jetzt die Funktion `Examine` nicht mehr richtig arbeitete. Sie übergab mir völlig unsinnige Längen – ich war dem Verzweifeln nahe, denn ich konnte den Fehler nicht lokalisieren.

Das Programm ist im Prinzip auch richtig, aber folgendes war passiert: C weist bei der Anforderung von Speicher, zum Beispiel durch

```
struct FileInfoBlock MyInfoBlock;
```

einen Bereich zu, dessen Startadresse durch zwei teilbar, also auf WORD geeicht ist. Dies liegt an der Prozessor-Struktur des 68000-Prozessors. Das DOS arbeitet aber wie oben schon gesagt etwas anders. Es erwartet, daß die Startadressen der Speicherbereiche, mit denen es arbeiten soll, durch vier teilbar ist, also auf LONG geeicht sind. Bei meinem Programm war scheinbar zufällig zu Beginn `&MyInfoBlock` durch vier teilbar – `Examine` arbeitete korrekt. Nach dem hinzufügen einer Variable war dies dann scheinbar nicht mehr der Fall – das DOS arbeitete nicht mehr korrekt. Das obige Programm sieht also nach der Korrektur folgendermaßen aus:

```
struct FileInfoBlock *MyInfoBlock;
long   MyLock,*Hilfe;
...

if((MyLock=Lock("test.prg",ACCESS_READ))==Ø)
   close_all(); /* Fehler -> Programm verlassen */
if((Hilfe=AllocMem(sizeof(FileInfoBlock)+4,MEMF_CLEAR))==Ø)
   close_all();
MyInfoBlock=Hilfe+Hilfe%4; /* durch vier teilbar!! */
if(Examine(MyLock,MyInfoBlock)==Ø)
   close_all();
printf("File-Länge: %d\n",MyInfoBlock->fib_Size);
...
```

Nun können wir also schon eine Datei untersuchen, wir wollen sie aber auch noch lesen oder in sie schreiben können. Bevor das möglich ist, müssen wir das File noch öffen:

```
MyFile = Open(FileName,Art);
  DØ             D1     D2
```

FileName gibt wieder den Namen des Files an. `MyFile` zeigt auf eine File-Struktur. Wie bei der Lock-Struktur kann uns deren Aufbau wieder völlig egal sein. Sie wird vom DOS aber bei den Funktionen `Read()` und `Write()` benötigt. Art gibt an, um was für einen Zugriff es sich handeln soll. Hier sind `MODE_OLDFILE` zum Öffnen eines alten und `MODE_NEWFILE` eines neuen Files möglich. Um

Schreiben und Lesen zu können muß die Datei mit `Open()` geöffnet werden, es ist nicht nötig, sich vorher ein Schloß übergeben zu lassen!

Nachdem wir die Datei fertig bearbeitet haben, müssen wir sie wieder schließen. Dies geschieht mit

```
Close(MyFile);
       D1
```

Wird beim Öffnen mit `Open()` eine Null übergeben, so war es dem DOS nicht möglich, die Datei zu öffnen. Diese Art der Fehlermeldung sollte aber mittlerweile jedem klar sein, denn sie ist für das Arbeiten auf dem Amiga typisch!

```
Anzahl = Read(MyFile,Adresse,Länge);
  DØ            D1      D2      D3
```

Liest aus der Datei, die durch `MyFile` spezifiziert ist, von der augenblicklichen Position Länge Bytes in den Speicher ab Adresse. Länge gibt also an, wie viele Bytes gelesen werden sollen und Adresse zeigt auf den Anfang eines Speicherbereichs, in den geschrieben werden soll. Die Funktion meldet mit Anzahl, wieviele Bytes wirklich gelesen werden konnten. Ist Anzahl gleich Null, so ist das Ende der Datei erreicht. Falls Anzahl kleiner Länge ist, so ist das File kürzer als wir es erwartet haben.

```
Anzahl = Write(MyFile,Adresse,Länge);
  DØ             D1      D2      D3
```

Schreibt die Länge der Bytes aus dem Speicherbereich ab Adresse in die durch `MyFile` spezifizierte Datei. Dabei muß unbedingt darauf geachtet werden, daß Adresse ein gerader Wert ist (siehe Probleme mit der FileInfoBlock-Struktur!). Diese drei Angaben sind analog wie bei der Funktion `Read()` anzusehen. `Write()` übergibt mit Anzahl entweder die Anzahl der Bytes, die wirklich geschrieben werden konnten bzw. `-1`, falls ein Fehler aufgetreten ist.

Zum Abschluß der kurzen Exkursion ins DOS noch eine weitere Funktion:

```
position = Seek
   DØ
            (MyFile,neue_position,Art);
              D1          D2        D3
```

Diese Funktion macht es möglich, festzulegen, ab welcher Position in der Datei gelesen bzw. geschrieben werden soll. Man braucht sie beispielsweise, um nach dem Schreiben eines gepackten Bildes, die Länge des Files bzw. des Bodys, in die Chunk-Header am Anfang des Files einzutragen. `MyFile` spezifiziert wieder die Datei. `neue_position` gibt an, auf welches Byte DOS neu positionieren soll. Mit Art wird angegeben, wie dieser Wert interpretiert wird. Hier sind folgende Flags möglich:

`OFFSET_BEGINNING`	Ab Anfang der Datei
`OFFSET_END`	Ab Ende der Datei
`OFFSET_CURRENT`	Ausgehend von der augenblicklichen Position

Mit `position` wird dem aufrufenden Programm die neue Stellung des Zeigers (Stelle der Positionierung) im File übergeben. Falls ein Fehler aufgetreten ist (zum Beispiel versucht wurde, vor den Dateianfang bzw. hinter das Dateiende zu positionieren), wird `-1` gemeldet.
Wollen wir also zum Beispiel auf das vierte Byte der Datei positionieren, so rufen wir die Funktion folgendermaßen auf.

```
if(Seek(MyFile,4,OFFSET_BEGINNING)!=-1)
   /* ok! */
else
   /* Fehler! */
```

Schreiben wir jetzt zum Beispiel mit (`LONG lenfile`)

```
Write(file,&lenfile,4);
```

vier Bytes in die Datei, so wird in sie ab dem vierten Byte der LONG-Wert `lenfile` geschrieben. Es ist klar, daß nicht eingefügt, sondern überschrieben wird!

8.4 Kurze Zusammenfassung aller wichtigen Funktionen

Intuition

`IntuitionBase:`	Zeiger auf die IntuitionBase
`MyNewScreen:`	NewScreen-Struktur
`MyNewWindow:`	NewWindow-Struktur
`MyWindow:`	Zeiger auf eine Window-Struktur
`MyScreen:`	Zeiger auf eine Screen-Struktur
`ergebnis:`	0-> Funktion meldet Fehler / 1 -> ok
`Menu1:`	Erstes Menü in der Menüliste
`MNum:`	16-Bitwert, der ein Menü, Item oder Subitem bezeichnet

```
IntuitionBase=OpenLibrary
              ("Intuition.library",Ø)
CloseLibrary(IntuitionBase);
```

Screen-Funktionen

```
MyScreen=OpenScreen(&MyNewScreen);
                          AØ

CloseScreen(MyScreen);
              AØ

ShowTitle(Screen,bool);
            AØ    DØ

MoveScreen(MyScreen,dx,dy);
              AØ   DØ D1

ScreenToBack(MyScreen);
ScreenToFront(MyScreen);
              AØ

ergebnis=OpenWorkBench();

ergebnis=CloseWorkBench();

WBenchToFront();
WBenchToBack();

DisplayBeep(MyScreen);
```

Window-Funktionen

```
MyWindow=OpenWindow(&MyNewWindow);
                          AØ

CloseWindow(MyWindow);
              AØ

MoveWindow(MyWindow,dx,dy);
              AØ   DØ D1

WindowToFront(MyWindow);
WindowToBack(MyWindow);
              AØ

SizeWindow(MyWindow,dx,dy);
              AØ   DØ D1

SetWindowTitles
(MyWindow,WindowText,ScreenText);
     AØ        A1          A2

ReportMouse(MyWindow,bool);
```

Menü-Funktionen

```
SetMenuStrip(MyWindow,&Menü1);
                AØ       A1

ClearMenuStrip(MyWindow);
                  AØ

OnMenu(MyWindow,MNum);
OffMenu(MyWindow,MNum);
           AØ      DØ
MENUNUM(MNum);
ITEMNUM(MNum);
SUBNUM(MNum);
```

Grafikfunktionen

GfxBase	Zeiger auf die GfxBase-Struktur
MyRastPort	Zeiger auf eine RastPort-Struktur zum Beispiel eines Screens oder Windows MyRastPort=&(MyScreen→RastPort); MyRastPort=MyWindow→RPort;
MyViewPort	Zeiger auf eine ViewPort-Struktur MyViewPort=&(MyScreen→ViewPort);
Farbreg	Nummer eines Farbregisters
r,g,b	Werte zwischen 0 und 15, die die Helligkeiten des Rot-, Grün- und Blauanteils einer Farbe bestimmen
x,y	Pixelkoordinaten
Modus	Zeichenart (JAM1,JAM2, COMPLEMENT, INVERSEVID)

```
GfxBase=OpenLibrary("Graphic.library",Ø)
CloseLibrary(GfxBase);
```

```
SetRGB4(MyViewPort,Farbreg, r, g, b);
           AØ        DØ     D1 D2 D3

SetAPen(MyRastPort,Farbreg);
           A1        DØ

SetBPen(MyRastPort,Farbreg);
           A1        DØ

SetDrMd(MyRastPort,Modus);
           A1       DØ

Move(MyRastPort, x , y );
          A1    DØ  D1

Draw(MyRastPort, x , y );
          A1    DØ  D1

WritePixel(MyRastPort, x , y );
              A1      DØ  D1

PolyDraw(MyRastPort,Anzahl,Daten);
              A1     DØ     AØ

Text(MyRastPort,Text,Länge);
        A1      AØ    DØ

SetRast(MyRastPort,Farbreg);
           A1        DØ
```

DOS-Funktionen

mylock	Zeiger auf eine Lock-Struktur
FileName	Zeiger auf einen String, der den File-Namen angibt
Art	Siehe genaue Beschreibung der jeweiligen Funktion
file	Zeigt auf eine File-Struktur
MyInfoBlock	Zeigt auf eine FileInfoBlock-Struktur
succ	True bzw. False je nach Erfolg
anzahl	Anzahl der gelesenen bzw. geschriebenen Bytes
Länge	Anzahl der zu lesenden bzw. schreibenden Bytes

```
Adresse     Zeiger auf Speicherbereich
position    Position im File
            (analog neue_position.)

DosBase=OpenLibrary("dos.library",Ø)
CloseLibrary(DosBase);

mylock=Lock(FileName,Art);
  DØ            D1     D2

UnLock(lock);
        D1

succ=Examine(mylock,MyInfoBlock);
 DØ             D1       D2

file=Open(FileName,Art);
 DØ          D1       D2

Close(file);
       D1

position=Seek(file,neue_position,Art);
   DØ           D1          D2        D3

anzahl=Read(file,Adresse,Länge);
  DØ          D1     D2      D3

anzahl=Write(file,Adresse,Länge);
  DØ          D1     D2      D3
```

8.5 Das IFF-Format - alles will richtig aufgehoben sein

Dieser Abschnitt ist schon in vielen Veröffentlichungen behandelt worden. Trotzdem ist es meiner Meinung nach nötig, dieses Format zur Bildspeicherung zu erklären, denn meist ließ die Vollständigkeit zu wünschen übrig. Die Dokumentation des IFF-Formats nimmt im Original mehr als 50 Seiten ein. Allein dieser Umfang legt nahe, sich nur auf das Wesentliche zu beschränken. Jeder, der sich mehr mit der Definition dieser Formate beschäftigen will, sei hier auf die entsprechende Literatur im Anhang verwiesen. Die Abkürzung IFF steht für Interchange File Format und ist ein Standard, der die Bearbeitung zum Beispiel von Bildern, Texten und Musik mit verschiedenen Programmen ermöglichen soll.

Das IFF-Format wurde von dem Software-Hersteller Electronic Arts vorgeschlagen, und heute verwenden es alle professionellen Programme auf dem Amiga. Dieses Format ist aber nicht nur für den Amiga, sondern universell auch für andere Rechner zu gebrauchen und findet dort Einsatz. Wie notwendig eine Einführung eines Standards war, versteht jeder, der zum Beispiel mit viele Mühe mit einem Malprogramm ein Bild gezeichnet und versucht hat, dies in einem andereren Programm zu verwenden, um dann festzustellen, daß dies leider nicht fähig ist, das Gemälde zu verarbeiten. Außerdem steht man als Programmierer immer vor dem Problem, mit welchem Format man es eigentlich zu tun hat und wie in aller Welt dies aufgebaut ist. Ein verallgemeinertes Format für die verschiedensten Anwendungen bedingt natürlich, daß nicht unbedingt die höchste Packungsdichte zu erreichen ist. Auch ist darauf zu achten, daß es immer nach oben kompatibel bleibt. Erfreulicherweise hat sich der gut durchdachte Vorschlag von ECA (Electronic Arts) durchgesetzt und bringt sicher sowohl den Anwendern, als auch den Software-Entwicklern auf dem Amiga viele Vorteile. Die Philosophie des IFF-Standards: Lieber ein wenig Mehraufwand beim Laden und Speichern der Daten, als unzählige verschiedene Formate zu berücksichtigen.

Für uns interessant ist an dieser Stelle natürlich nur die Verwendung des IFF-Formats zum Speichern der von uns berechneten Bilder. So ist es auch möglich mit anderen

Grafikprogrammen (zum Beispiel Deluxe Paint oder Aegis Images) unsere Bilder zu laden und nachträglich zu verändern.

Um die bereits angesprochene Aufwärtskompatibilität zu sichern, wird mit sogenannten Chunks gearbeitet. (Chunk übersetzt man am besten mit Kennstück.)

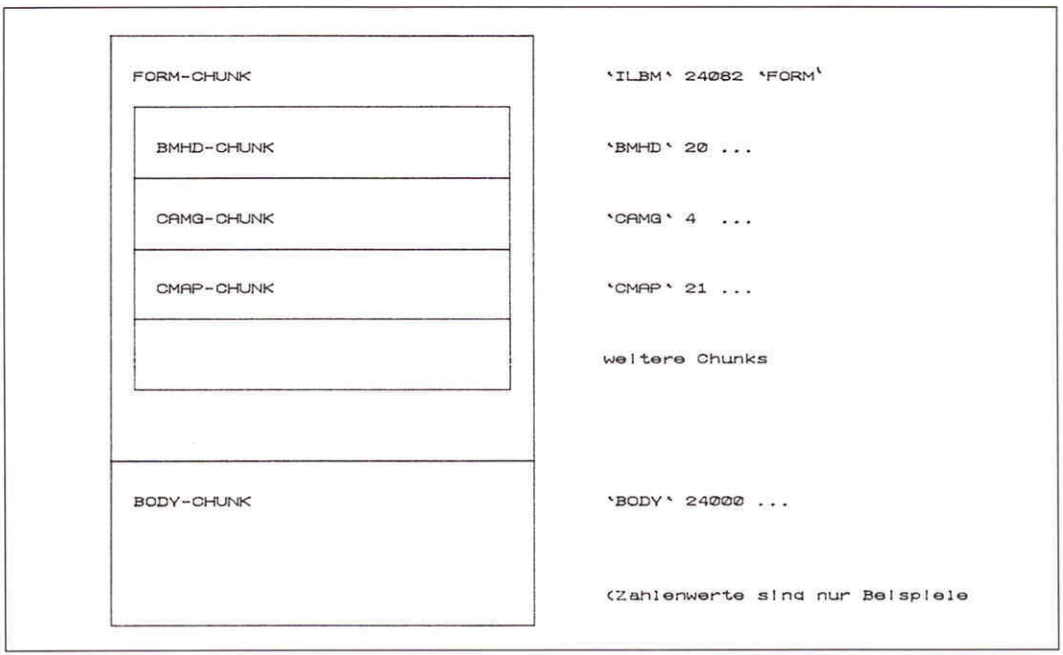
Am Anfang eines Chunks steht immer ein vier Byte langes Codewort, das den Typ der nachfolgenden Daten angibt. Unter einem Chunk muß man sich einen einzelnen Datenblock vorstellen, der eine bestimmte Art von Information (zum Beispiel das Format des Bildes) enthält. Alle Chunks zusammen enthalten dann die notwendigen Informationen und bilden das endgültige File (siehe Abbildung 8-4). Ein Chunk ist folgendermaßen aufgebaut:

```
Chunk-Identifikation (4 Bytes)      (z.B. CMAP)
Chunk-Länge          (4 Bytes)
Daten (Anzahl gemäß der Chunk-Länge.)
```

Ein IFF-File beginnt immer mit dem sogenannten FORM-Chunk. Dieser enthält die Information, daß dies ein IFF-File ist, wie lang das File ist, und um welche Art von Daten es sich handelt.

Aufbau des FORM-Chunks:

```
(4 Bytes)      F,O,R,M
(Langwert)     Filelänge-8
(4-Bytes)      Kennung der Dateiart
               FTXT für eine Textdatei
               8SVX für digitalisierten Sound
               SMUS für Musik-Daten
               ILBM für Bilder
```



8-4 Aufbau eines IFF-Bildes

Wir wollen uns hier auf ILBM-Files beschränken. ILBM steht für Interleaved Bitmap und gibt an, daß das File Bilddaten enthält. Wie jeder C-Programmierer auf dem Amiga weiß, ist eine Fülle von Informationen nötig, um ein Screen-Format richtig zu definieren. Um außerdem möglichst vielseitig zu bleiben, enthalten einige Chunks Daten, die wir nur zum Bildspeichern gar nicht ausnutzen können oder wollen. Deshalb will ich mich hier auf die Erklärung der zum Bildspeichern notwendigen Chunks und Chunk-Elemente beschränken.

Nachdem wir nun wissen, daß wir ein IFF-Bild-File vor uns haben und dessen Länge kennen, können wir die Analyse der folgenden Chunks vornehmen.

Der BMHD-Chunk

BMHD-Chunk steht für BitMapHeader-Chunk. Dieser enthält die notwendigen Informationen über Format, Auflösung, usw. des Bildes. An dieser Stelle ist zu sagen, daß ein bestimmter Chunk nicht unbedingt vorhanden sein muß, es sei denn, er enthält »lebensnotwendige« Informationen. Bei einer Bild-Datei darf zum Beispiel der CMAP-Chunk fehlen (er enthält die Farben des Bildes), aber der BMHD-Chunk muß verständlicherweise immer vorhanden sein. Er hat folgenden Aufbau:

```
(4 Bytes)    Chunkkennung      B,M,H,D
(Langwort)   Chunklänge (HØxØØ14 -> 2Ø)
(Wort)       Breite des Bildes in Pixeln (durch 8 teibar!)
(Wort)       Höhe des Bildes in Pixeln (durch 8 teilbar!)
(Wort)       x-Position des Bildes (auch negativ)
(Wort)       y-Position des Bildes (auch negativ)
(Byte)       Anzahl der Bitplanes
(Byte)       Masking
(Byte)       Kompression
(Byte)       Ø (für Erweiterungen)
(Wort)       transparente Farbe (Farbregister)
(Byte)       x-Aspekt des Bildes
(Byte)       y-Aspekt des Bildes
(Wort)       Page-Breite (vorzeichenbehaftet!)
(Wort)       Page-Höhe (vorzeichenbehaftet!)
```

Die Chunk-Länge gibt an, wie viele Daten-Bytes noch folgen. Die ersten acht Byte des Chunks werden also nicht mitgerechnet. Bei den Angaben der Anzahl der Pixel ist darauf zu achten, daß sie durch 8 teilbar ist, sonst kann das Bild nicht geladen werden. Ein Bild ist also nur auf ein Byte genau positionier- und definierbar. Ob das Bild gepackt ist, oder nicht, gibt das »Komprimiert«-Byte an. Steht hier der Wert 1, so ist es komprimiert. Wie das Packen funktioniert, steht weiter unten bei der Erklärung des BODY-Chunks. Die Werte x-Aspekt und y-Aspekt geben das Verhältnis von Bildbreite zur Bildhöhe an. Haben wir zum Beispiel ein Format von 320 x 200 Bildpunkten, so müssen hier die Werte 10 und 11 stehen.

Das »Masking«-Byte gibt an, welche Maskierungsart benutzt werden soll. Der Wert Null bedeutet kein Masking. Steht hier eine Eins, so enthalten die Bilddaten eine Maske. Eine Zwei bedeutet, daß an Stelle der transparent definierten Farbe der Hintergrund zu sehen ist. Noch eine weitere Art der Maskierung ist möglich. Steht hier der Wert 3, so wird ein ein Pixel breiter Rand um das Bild gezogen und nach innen solange aufgefüllt, bis man auf eine andere Farbe trifft. In die so gefüllte Fläche wird der Hintergrund eingeblendet.

Der BMHD-Chunk enthält natürlich noch nicht alle notwendigen Daten. Für spezielle Anwendungen wie Sprite-Verarbeitung und Animation gibt es noch einige Chunks, deren Erklärung zuweit führen würde. Wie gesagt –

für uns steht das Ziel im Vordergrund, ein von uns berechnetes Bild im IFF-Format abspeichern und laden zu können. Weitere wichtige Informationen enthält der bereits oben erwähnte CMAP-Chunk (ColorMap-Chunk). Dieser Datenblock beinhaltet die Farbinformationen des Bildes und hat folgenden Aufbau:

```
(4 Bytes)    C,M,A,P
(Langwort)   Chunk-Länge (hängt von der Farbanzahl ab.)
(Byte)       FarbeØ -> rot
(Byte)       FarbeØ -> grün
(Byte)       FarbeØ -> blau
(Byte)       Farbe1 -> rot
(Byte)       Farbe1 -> grün
(Byte)       Farbe1 -> blau
...
```

Die Länge des CMAP-Chunks hängt von der Anzahl der im Bild vorhandenen Farben ab (sofern nicht einfach immer 32 Farbregister abgespeichert werden). Nach der Chunk-Identifikation und Länge folgen unmittelbar die RGB-Werte der einzelnen Farbregister als Bytes (natürlich nicht vorzeichenbehaftet!). Setzt man nun diese Farben mit dem Befehl `SetRGB4(RastPort,i,r,g,b)`, so muß man beachten, daß man die Werte `r`, `b` und `g` erhält, indem die eingelesenen Werte rot, grün und blau durch 16 dividiert werden.

Schon etwas exotischer wird es bei dem CRNG-Chunk (ColorRegisterRange-Chunk). Er enthält Angaben über die Farbregister und die Geschwindigkeit mit der Farbanimation in diesem Bild stattfinden soll.

Bei Deluxe-Paint dient der erste Chunk zum Festlegen der Farbregister, die für die Shade-Funktion verwendet werden sollen, die restlichen drei enthalten die Daten für die Farbanimation.

```
(4 Bytes)    C,R,N,G
(Langwort)   Chunk-Länge (hier steht eine 8)
(Wort)       Nullwort (Ø)
(Wort)       Geschwindigkeit
(Wort)       Aktiviert?
(Byte)       unteres Farbregister
(Byte)       oberes Farbregister
```

Nach der Angabe der Chunklänge folgt ein Nullwort, daß für zukünftige Erweiterungen gedacht ist. Hier sollte also eine Null stehen.

Die Geschwindigkeit mit der die Animation, das heißt das Durchlaufen der Farben, ablaufen soll, wird als Wort angegeben. Dabei bedeutet der Wert 16384 sechzig Schritte pro Sekunde. Steht hier ein kleinerer Wert, so sinkt die Geschwindigkeit. Steht in dem nun folgenden Wort ein Wert ungleich Null, so wird die Animation unmittelbar nach dem Laden gestartet. Das obere und das untere Farbregister folgen nun noch als Byte-Werte.
Den CAMG-Chunk gibt es speziell für den Amiga. Er enthält als Langwort den `viewport mode` des Bildes. Dieser Wert kann direkt in die NewScreen-Struktur übernommen werden. Er macht es möglich, den Dual-Playfield oder den Hold-And-Modify Modus für das Bild festzulegen.

Nun fehlt nur noch das Herzstück des Bildes, die eigentlichen Bildaten. Nach Angabe aller notwendigen bzw. gewünschten Chunks, folgt der sogenannte BODY-Chunk, also das eigentliche Bild. Wie bereits beim BMHD-Chunk angesprochen, gibt es zwei Arten der Bildspeicherung.

Ist das »Komprimiert«-Byte im BMHD-Chunk gleich Null sind die Daten nicht komprimiert. Nach der Chunk-Kennung `B,O,D,Y` folgen, und das ist wichtig, nacheinander je eine Zeile der einzelnen Bit-Maps. Das Bild

ist also Zeile für Zeile abgespeichert. Da Disketten teuer sind, liegt es nahe zu versuchen, die Bilddaten zu komprimieren. Daß dies auch möglich ist, zeigen die großen einfarbigen Flächen, die fast jedes Bild enthält. Deshalb wird folgende Verschlüsselung angewendet: Vor den Daten-Bytes steht ein Befehls-Byte, das angibt, wie diese ausgewertet werden sollen. Dies Befehls-Byte (n) ist als vorzeichenbehaftet anzusehen. Steht hier ein Wert n zwischen -1 und -127, so wird das folgende Byte (-n+1) mal wiederholt. Ein Wert größer oder gleich Null bedeutet, daß die nächsten (n-1) Bytes direkt geladen werden sollen. Eine -128 bedeutet NOP; es geschieht gar nichts und das nachfolgende Byte ist wieder als Befehls-Byte anzusehen. Wichtig ist, daß dieses Komprimieren auch zeilenweise geschieht. Also auch hier stehen nacheinander komprimiert die einzelnen Zeilen der Bit-Planes.

Noch etwas Wichtiges: Die Länge des gesamten Files muß gerade sein. Sollte sich nach dem Speichern des Body-Chunks eine ungerade Länge ergeben, so ist einfach eine Null anzuhängen. Diese muß bei der Längenangabe beim Form-Chunk berücksichtigt werden, bleibt aber für die Länge des Body-Chunks ohne Belang.

Der Aufbau des IFF-Formats ist, wie jetzt anschaulich geworden, sehr universell angelegt. Das Arbeiten mit Chunks macht es möglich, daß jedes Programm eigene Chunks anlegen kann, die spezielle Daten enthalten und trotzdem für andere Programme lesbar bleiben. Die bis jetzt erklärten Chunks sind für unsere Anwendung genug, aber schauen wir uns doch noch als Beispiel den Anfang eines Bild-Files an, das mit dem Programm Deluxe-Paint abgespeichert wurde.

```
ØØØØ: 464F524D ØØØ04B46 494C424D 424D4844    FORM..KFILBMBMHD
ØØ1Ø: ØØØØØØ14 Ø14ØØØC8 ØØØØØØØØ Ø5Ø2Ø1ØØ    .....@.........
ØØ2Ø: ØØØØØAØB Ø14ØØØC8 434D415Ø ØØØØØØ6Ø    .....@..CMAP...'
ØØ3Ø: ØØØØØØ7Ø ØØØØBØØØ ØØFØØØ1Ø FØ5Ø5ØØØ    ...p.........PP.
ØØ4Ø: ØØ7ØØØØØ BØØØØØFØ 4Ø8ØDØBØ AØØØDØCØ    .p.......@.......
ØØ5Ø: ØØFØFØØØ FØFØ6ØFØ 6ØØØBØ4Ø ØØ1Ø8ØØØ    ......'.'..@....
ØØ6Ø: 1Ø5ØØØ1Ø 1Ø1Ø2Ø2Ø 2Ø3Ø3Ø3Ø 4Ø4Ø4Ø5Ø    .P....  ØØØ@@@P
ØØ7Ø: 5Ø5Ø6Ø6Ø 6Ø7Ø7Ø7Ø 8Ø8Ø8Ø9Ø 9Ø9ØAØAØ    PP'''ppp........
ØØ8Ø: AØBØBØBØ CØCØCØDØ DØDØEØEØ EØFØFØFØ    ................
ØØ9Ø: 43524E47 ØØØØØØØ8 ØØØØØØØØ ØØØ1141F    CRNG............
ØØAØ: 43524E47 ØØØØØØØ8 ØØØØØAAA ØØØ1Ø3Ø7    CRNG............
ØØBØ: 43524E47 ØØØØØØØ8 ØØØØØAAA ØØØ1ØØØØ    CRNG............
ØØCØ: 43524E47 ØØØØØØØ8 ØØØØØAAA ØØØ1ØØØØ    CRNG............
ØØDØ: 424F4459 ØØØØ4A76 D9ØØD9ØØ D9ØØD9ØØ    BODY..Jv........
ØØEØ: D9ØØD9ØØ D9ØØD9ØØ D9ØØD9ØØ D9ØØD9ØØ    ................
ØØFØ: D9ØØD9ØØ D9ØØD9ØØ D9ØØD9ØØ D9ØØD9ØØ    ................
Ø1ØØ: D9ØØD9ØØ D9ØØD9ØØ D9ØØD9ØØ D9ØØD9ØØ    ................
Ø11Ø: D9ØØD9ØØ D9ØØD9ØØ D9ØØD9ØØ D9ØØD9ØØ    ................
Ø12Ø: D9ØØD9ØØ D9ØØD9ØØ D9ØØD9ØØ D9ØØD9ØØ    ................
.................................
(Die PaintCan von Deluxe Paint mit type ... opt h ausgedruckt.)
```

Am Anfang steht wie zu erwarten die Kennung des IFF-Formats FORM. Der folgende Chunk gibt die Länge des gesamten Files an, hier also H x 00004B 46 Bytes. Außerdem handelt es sich um ein ILBM-File, wie an der Kennung zu sehen ist. Es folgt der BMHD-Chunk, dann der CMAP-Chunk, vier CRNG-Chunks (typisch für Deluxe-Paint) und an Schluß der Body-Chunk. Es empfiehlt sich zum besseren Verständnis einmal die Angaben der einzelnen Chunks zu entschlüsseln und zu überprüfen.

Nach all der Theorie wieder ein wenig Praxis! Es folgt ein Programmpaket, das alle Funktionen zum Laden und Speichern von IFF-Bildern enthält. Die Übergabe der Parameter und der Funktionen sind im Programm selbsterklärend.

Will man die Funktionen in eigenen Programmen verwenden, so wird das kompilierte Objekt-Modul des Programmpakets zu dem eigenen Programm »dazugelinkt«. Das hat den Vorteil, daß nicht immer wieder in den Programmen die gleichen Routinen auftauchen.

Beim Aztec-C-Compiler geschieht dies auf folgende Weise: Kompilieren des Programmpakets *iff.c*. Das entstehende Objekt-Modul *iff.o* wird weiterverwendet. Beim Linken tritt ein Fehler auf, denn das Programm enthält keine `main()`-Funktion. Um nun die Funktion `save()` oder `load()` zu verwenden, müssen diese im eigenen Programm als externe Funktionen definiert werden, also `extern short save();` bzw. `extern short load();`. Das Anbinden der Routinen geschieht nun beim Linken des eigenen Programms (*MyFile*), also

```
'ln MyFile.o iff.o m.lib c.lib -o Myfile.run'.
```

Das Vorgehen beim Lattice-C-Compiler ist analog. Nach dem Kompilieren erhalten wir wieder ein Objekt-File *iff.o*. Dies wird dann beim Linken an ein eigenes Programmodul wie folgt eingebunden:

```
'blink FROM LIB:c.o+MyFile.o TO MyFile
iff.o+LIB:lcm.lib+LIB:lc.lib+LIB:amiga.lib'
```

Programmodul zum Laden und Speichern von Bildern im IFF-Format:

```
1  /*************************************************************************/
2  /* iff.c                                                  26.Ø1.88  */
3  /* Version 3.Ø                                                      */
4  /*                                                                  */
5  /*                                                                  */
6  /* Lade- und Speicherroutine fuer Bilder im IFF-Format              */
7  /*                                                                  */
8  /* fuer Markt+Technik geschrieben von Heiko Knappe                  */
9  /*                                                                  */
1Ø /*                                                                  */
11 /* Die Routinen sind so geschrieben, dass sie ohne Schwierigkeiten in ein */
12 /* groesseres Programm eingebunden  werden koennen.  Auch bei der Auswer- */
13 /* tung  anderer  IFF-Formate  sind sie  sicher hilfreich.  Geladen  und */
14 /* gespeichert  werden  koennen  sowohl 'uncompressed' (z.B.Graphicraft), */
15 /* als auch 'compressed' (z.B. Deluxe-Paint) Formate.  Die  verschiedenen */
16 /* Aufloesungen und Display-Modes werden berueckSichtigt! Auch HAM-Bilder */
17 /* koennen geladen und gespeichert werden werden!                   */
18 /* Der neu geoeffnete Screen bleibt beim Laden dunkel (bei HAM natuerlich */
19 /* nicht).  Die Farben werden erst nach Enden des Diskzugriffs aufgeblen- */
2Ø /* det.                                                             */
21 /*************************************************************************/
22
23
24 #include <exec/types.h>
25 #include <exec/memory.h>
26 #include <intuition/intuition.h>
27 #include <libraries/dos.h>
28
29
3Ø #define BMHDChunk 1
31 #define CMAPChunk 2
32 #define BODYChunk 3
33 #define CAMGChunk 4
34 #define Unknown   5
35
36 #define MEM_SIZE 2ØØØØ /* Buffer fuer Speichern                  */
37
38 struct CMAPC          /* Definition des CMAP-Chunks             */
39 {
4Ø  int anz;            /* Anzahl der Farben                      */
41  struct ColorRegister /* Definition einer Farbe                 */
42  {
```

```
43    UBYTE red;
44    UBYTE green;
45    UBYTE blue;
46   } Color[64];          /* lieber zuviel als zuwenig              */
47  };
48
49  struct FORMC           /* Definition des Anfang-Chunks           */
50  {
51   char  FORMName[4];    /* handelt es sich um ein IFF-Format?     */
52   long  len;            /* Gesamte Laenge des Files               */
53   char  ILBM[4];        /* handelt es sich um ILBM-Format?        */
54  };
55
56  struct BMHDC           /* Definition des BMHD                    */
57  {
58   char  BMHDName[4];
59   long  lenBMHD;        /* Laenge des BMHD-Chunks                 */
60   short Width;          /* Breite des Bildes                      */
61   short Height;         /* Hoehe des Bildes                       */
62   short xpos;           /* x-Position des Bildes                  */
63   short ypos;           /* y-Position des Bildes                  */
64   char  Depth;          /* Anzahl der BitPlanes                   */
65   char  mask;           /* Mask-Bit                               */
66   char  compression;    /* Bild 'compressed'?                     */
67   char  null;
68   short color;
69   char  xaspect;
70   char  yaspect;
71   short PWidth;
72   short PHeight;
73  };
74
75  struct CHUNKID         /* hier werden die zu analysierenden Bytes */
76  {                      /* abgelegt                               */
77   char  Name[4];        /* Name des Chunks                        */
78   int   len;            /* Laenge des Chunks                      */
79  };
80
81  struct HEADER          /* diese Struktur enthaelt die Zeichenfolge */
82  {                      /* um den Chunk zu erkennen.              */
83   char bmhd[4];
84   char cmap[4];
85   char body[4];
86   char camg[4];
87  };
88
89  struct CAMGC           /* Definition des CAMG */
90  {
91   char CAMGName[4];
92   long lenCAMG;
93   long ViewModes;
94  };
95
96  struct CMAPC CMAP1;
97  struct FORMC FORM1;
98  struct BMHDC BMHD1;
99  struct CHUNKID CHUNKid;
100 struct FORMC FORMs = {'F','O','R','M',0,'I','L','B','M'};
101 struct BMHDC BMHDs = {'B','M','H','D',20,0,0,0,0,0,2,0,0,0,10,11,0,0};
102 struct CAMGC CAMGs = {'C','A','M','G',4,0};
103 struct HEADER Header=
104 {
105  'B','M','H','D',    /* BMHD-Chunk */
106  'C','M','A','P',    /* CMAP-Chunk */
107  'B','O','D','Y',    /* BODY-Chunk */
108  'C','A','M','G',    /* CAMG-Chunk */
109 };
110
111 char cmap[]="CMAP";
112 char body[]="BODY";
113 char form[]="FORM";
114 char ilbm[]="ILBM";
115
116 struct IntuitionBase *IntuitionBase;
117 struct GfxBase *GfxBase;
118 struct ViewPort *MyViewPort;
119 struct RastPort *MyRastPort;
120 struct BitMap *MyBitMap;
121 struct Screen *myscr;
122 struct NewScreen mynewscr=
123  {0,0,320,200,2,0,1,0,CUSTOMSCREEN,0,0,0,0};
124
125 long file,end,*waste,laenge;
126 BYTE *plane[6],*speicher,*zeiger,*zeiger0;
```

```
127
128
129
130 /*********************/
131 /* Laderoutinen      */
132 /*********************/
133
134 /* Aufblenden der Farben */
135 blendup()
136 {
137  /* Bild geladen -> Farben setzen */
138  static float r[64],g[64],b[64];
139  static int i,j;
140
141  for(i=Ø;i<64;++i)
142  {
143     r[i]=(CMAP1.Color[i].red)/16;    /* Aufblenden in 16 Schritten */
144     g[i]=(CMAP1.Color[i].green)/16;
145     b[i]=(CMAP1.Color[i].blue)/16;
146  }
147  for(j=1;j<=16;++j)
148     for(i=Ø;i<CMAP1.anz;++i)
149        SetRGB4(MyViewPort,i,(int)(r[i]*j/16),
150                (int)(g[i]*j/16),(int)(b[i]*j/16));
151 }
152
153
154 /* Analyse des Headers */
155 /* fname: zeigt auf den Namen des Bild-Files                    */
156 /* scr: zeigt auf den Screen in den das Bild geladen werden soll.  */
157 /*      Steht hier eine Null, so wird ein neuer Screen geoeffnet.  */
158 /*      Dieser bleibt dann solange offen, bis dieser herunterge-   */
159 /*      zogen wird. Das Programm kehrt erst dann zurueck. Tritt    */
160 /*      beim Laden kein Fehler auf, so wird eine Null uebergeben.  */
161 /*      folgende Fehler sind moeglich:                             */
162 /*        Fehler 1: File kann nicht geoeffnet werden.              */
163 /*        Fehler 2: kein IFF-File.                                 */
164 /*        Fehler 3: Screen bzw. Libs koennen nicht geoeffnet werden. */
165 load(fname,scr)
166 char *fname;
167 struct Screen *scr;
168 {
169  static short id;
170  static BYTE wait=1;
171  static int i;
172
173  if(scr!=Ø)
174  {
175     myscr=scr;
176     wait=Ø;
177  }
178  if(init_load_file(fname)==Ø)
179     return(1);
180  end=read_file(&FORM1,sizeof(FORM1));
181  if((strncmp(&FORM1.FORMName,&form,4)!=Ø)(strncmp(&FORM1.ILBM,&ilbm,4)!=Ø))
182  {
183    close_load_file();
184    return(2);
185  }
186  while(end!=Ø)
187  {
188     end=read_file(&CHUNKid,sizeof(CHUNKid));
189     if(end==8) id=test();
190     else {end=Ø; break;};
191     switch(id)        /* Welcher Chunk liegt vor? */
192     {
193     case BMHDChunk:
194            BMHD1.lenBMHD=CHUNKid.len;
195            end=read_file(&BMHD1.Width,CHUNKid.len);
196            if((end<CHUNKid.len)) break;
197            if(CHUNKid.len<(sizeof(BMHD1)-12))
198            {
199               close_load_file();
200               return(2);
201            }
202            mynewscr.Width    =BMHD1.Width;
203            mynewscr.Depth    =BMHD1.Depth;
204            mynewscr.Height   =BMHD1.Height;
205            mynewscr.LeftEdge =BMHD1.xpos;
206            mynewscr.TopEdge  =BMHD1.ypos;
207            if(mynewscr.ViewModes==Ø) /* ViewMode schon gesetzt? */
208            {
209               if(mynewscr.Width>32Ø) mynewscr.ViewModes+=HIRES;
210               if(mynewscr.Height>25Ø) mynewscr.ViewModes+=LACE;
```

```
                if(mynewscr.Depth==6) mynewscr.ViewModes+=HAM;
             }
             break;
      case CMAPChunk:
             CMAP1.anz=CHUNKid.len/3;
             for(i=Ø;i<CMAP1.anz;++i)
                end=read_file(&(CMAP1.Color[i].red),3);
             break;
      case BODYChunk:
             open_();
             get_pic();
             break;
      case CAMGChunk:
             end=read_file(&mynewscr.ViewModes,2);
             end=read_file(&mynewscr.ViewMo-
des,2); /* erst hier richtig */
             break;
      case Unknown:
             if((waste=AllocMem(CHUNKid.len,MEMF_CLEAR))==Ø)
               {end=Ø; break;}
             end=read_file(waste,CHUNKid.len);
             FreeMem(waste,CHUNKid.len);
             break;
      }
   }
   if(wait==1)
   {
      blendup();
      while(myscr->TopEdge<1);    /* Und, wie ist das Bild? */
      close_();
   }
   close_load_file();
   return(Ø);
}


/* Um welchen Chunk handelt es sich? */
test()
{
   static char *p;
   static short i;


   for(i=Ø,p=&Header;
       (i<sizeof(Header)/4)&&(strncmp(p,&CHUNKid.Name,4)!=Ø);++i,p+=4);
   return(i+1);
}


/* Laden der BitPlanes */
get_pic()
{
   static int i,j;
   static short count;
   static char *next,*nextØ,*n;

   if(BMHD1.compression!=Ø)          /* File compressed */
   {
      for(i=Ø;(i<BMHD1.Height)&&(end!=Ø);++i)
      {
         for(j=Ø;j<BMHD1.Depth;++j)
         {
            nextØ=next=plane[j]+BMHD1.Width/8*i;
            while((next-nextØ)<BMHD1.Width/8)
            {
               end=read_file(next,1);
               if(*next==-128) continue;  /* noop */
               if(*next>=Ø)
               {
                  count=1+(*next);
                  end=read_file(next,count);
                  next+=count;
                  continue;
               }
               if((*next<Ø)&&(*next!=-128))
               {
                  count=1-(*next);
                  end=read_file(next,1);
                  for(n=next;count>Ø;--count,*next=*n,++next);
                  continue;
               }
            }
```

```
291
292         }
293      }
294   }
295   else
296   {
297      for(i=Ø;i<BMHDl.Height;++i)
298         for(j=Ø;j<BMHDl.Depth;++j)
299            end=read_file(plane[j]+BMHDl.Width/8*i,BMHDl.Width/8);
3ØØ   }
3Ø1 }
3Ø2
3Ø3
3Ø4 /* Oeffnen des Screens bzw. der Libs */
3Ø5 open_()
3Ø6 {
3Ø7  static int i;
3Ø8
3Ø9  if(IntuitionBase==Ø)
31Ø     if((IntuitionBase=OpenLibrary("intuition.library",Ø))==Ø)
311     {
312        close_();
313        return(3);
314     }
315  if(GfxBase==Ø)
316     if((GfxBase=OpenLibrary("graphics.library",Ø))==Ø)
317     {
318        close_();
319        return(3);
32Ø     }
321  if(myscr==Ø)
322     if((myscr=OpenScreen(&mynewscr))==Ø)
323     {
324        close_();
325        return(3);
326     }
327  MyViewPort=&(myscr->ViewPort);
328  MyRastPort=&(myscr->RastPort);
329  for(i=Ø;i<CMAPl.anz;++i)    /* Screen zuerst dunkel */
33Ø     SetRGB4(MyViewPort,i,Ø,Ø,Ø);
331  MyBitMap=MyRastPort->BitMap;
332  for(i=Ø;i<BMHDl.Depth;++i)
333     plane[i]=MyBitMap->Planes[i];
334 }
335
336
337 /* Schliessen des Screens bzw. der Libs */
338 close_()
339 {
34Ø  if(IntuitionBase!=Ø) CloseLibrary(IntuitionBase);
341  if(GfxBase!=Ø)       CloseLibrary(GfxBase);
342  if(myscr!=Ø)      CloseScreen(myscr);
343  IntuitionBase=GfxBase=myscr=Ø;
344 }
345
346 init_load_file(fname)
347 char *fname;
348 {
349  if((laenge=filelaenge_(fname))==Ø)
35Ø     return(Ø);
351  if((file=Open(fname,MODE_OLDFILE))==Ø)
352     return(Ø);
353  if((speicher=AllocMem(laenge,MEMF_CLEAR))==Ø)
354     return(Ø);
355  Read(file,speicher,laenge);
356  zeiger=zeigerØ=speicher;
357  return(1);
358 }
359
36Ø filelaenge_(fname)
361 char *fname;
362 {
363  static struct FileInfoBlock *MyInfoBlock;
364  static long *MyLock,help,l;
365
366  if((MyLock=Lock(fname,ACCESS_READ))==Ø)
367     return(Ø);
368  if((help=AllocMem(1ØØØ,MEMF_CLEAR))==Ø)
369     return(Ø);
37Ø  MyInfoBlock=help+help%4;
371  if((Examine(MyLock,MyInfoBlock))==Ø)
372  {
```

```
373      UnLock(MyLock);
374      FreeMem(help,1ØØØ);
375      return(Ø);
376    }
377    UnLock(MyLock);
378    l=MyInfoBlock->fib_Size;
379    FreeMem(help,1ØØØ);
38Ø    return(l);
381  }
382
383  close_load_file()
384  {
385    if(speicher!=Ø) FreeMem(speicher,laenge);
386    if(file!=Ø) Close(file);
387  }
388
389  read_file(p,anz)
39Ø  BYTE *p;
391  int anz;
392  {
393    while((zeiger<=speicher+laenge)&&(zeiger-zeigerØ<anz))
394    {
395      *p=*zeiger;
396      ++zeiger,++p;
397    }
398    zeigerØ=zeiger;
399  }
4ØØ
4Ø1
4Ø2
4Ø3  /*********************/
4Ø4  /* Speicherroutine   */
4Ø5  /*********************/
4Ø6
4Ø7  /* Speichern von IFF-Bildern */
4Ø8  /* fname: zeigt auf den Namen des Bild-files                    */
4Ø9  /* modeh: gibt an, ob das Bild komprimiert abgespeichert werden soll.*/
41Ø  /*        es bedeutet:      Ø -> 'uncompressed'                 */
411  /*                          1 -> 'compressed'                   */
412  /* MyScreenh: zeigt auf den Screen des Bildes                   */
413  /* Ist das Speichern erfolgreich, so kehrt die
        Routine mit TRUE zurueck.                                    */
414    /* Andernfalls wird ein FALSE uebergeben.                     */
415  save(fname,modeh,MyScreenh)
416  char *fname;
417  BYTE modeh;
418  struct Screen *MyScreenh;
419  {
42Ø    static long i,j,k,anzcol,lenheader,lenfile,lencmap,lenbody;
421    static WORD color;
422    static BYTE ccount,*b1,*b2,*help;
423    static BYTE r,g,b,*plane[6],null=Ø;
424    static struct RastPort *MyRastPorth;
425    static struct BitMap *MyBitMaph;
426    static struct ViewPort *MyViewPorth;
427
428    if(MyScreenh==Ø) return(FALSE);
429    file=Open(fname,MODE_NEWFILE);
43Ø    if(file==Ø)
431      return(FALSE);
432    if((speicher=AllocMem(MEM_SIZE,MEMF_CLEAR))==Ø)
433    {
434      Close(file);
435      return(FALSE);
436    }
437    zeigerØ=zeiger=speicher;
438    MyRastPorth=&(MyScreenh->RastPort);
439    MyBitMaph=MyRastPorth->BitMap;
44Ø    MyViewPorth=&(MyScreenh->ViewPort);
441    /* Initialisierung des BMHD-Chunks */
442    BMHDs.Height=MyBitMaph->Rows;
443    BMHDs.Width=(MyBitMaph->BytesPerRow)*8;
444    BMHDs.Depth=MyBitMaph->Depth;
445    BMHDs.compression=modeh;
446    BMHDs.PWidth=BMHDs.Width;
447    BMHDs.PHeight=BMHDs.Height;
448    if(BMHDs.Height>=4ØØ) BMHDs.yaspect*=2;
449    if(BMHDs.Width>=64Ø) BMHDs.xaspect*=2;
45Ø    CAMGs.ViewModes=MyViewPorth->Modes;
451    for(i=Ø;i<BMHDs.Depth;++i)
452      plane[i]=MyBitMaph->Planes[i];
453    /* Anzahl der Farben? */
454    j=BMHDs.Depth;
455    if(j>5) j=5;  /* maximal 32 Farbregister abspeichern! */
```

```
456  for(i=Ø,anzcol=1;i<j;anzcol*=2,++i);
457  lencmap=anzcol*3;
458  lenbody=BMHDs.Width/8*BMHDs.Height*BMHDs.Depth;
459  lenheader=sizeof(FORMs)-8+sizeof(BMHDs)+sizeof(CAMGs)+lencmap+8+8;
46Ø  lenfile=lenheader+lenbody;
461  FORMs.len=lenfile;
462  /* Speichern des FORM-Chunks */
463  write_file(&FORMs,sizeof(FORMs));
464  /* Speichern des BMHD-Chunks */
465  write_file(&BMHDs,sizeof(BMHDs));
466  /* Speichern des CAMG-Chunks */
467  write_file(&CAMGs,sizeof(CAMGs));
468  /* Speichern des CMAP-Chunks */
469  write_file(&cmap,4);
47Ø  write_file(&lencmap,4);
471  lencmap+=8;
472  for(i=Ø;i<anzcol;++i)
473  {
474     color=GetRGB4(MyScreenh->ViewPort.ColorMap,i);
475     r=color/256; r*=16; color-=r*16;
476     g=color/16; g*=16; color-=g;
477     b=color*16;
478     write_file(&r,1);
479     write_file(&g,1);
48Ø     write_file(&b,1);
481  }
482  /* Speichern des BODY-Chunks */
483  write_file(&body,4);
484  write_file(&lenbody,4);
485  if(modeh==Ø)          /* nicht komprimiert! */
486  {
487     for(i=Ø;i<BMHDs.Height;++i)
488        for(j=Ø;j<BMHDs.Depth;++j)
489           write_file(plane[j]+BMHDs.Width/8*i,BMHDs.Width/8);
49Ø  }
491  else  /* komprimiert speichern! */
492  {
493     lenbody=Ø;
494     for(i=Ø;i<BMHDs.Height;++i)
495     {
496        for(j=Ø;j<BMHDs.Depth;++j)
497        {
498           for(ccount=Ø,k=Ø;k<(BMHDs.Width/8-1);++k)
499           {
5ØØ              b1=plane[j]+BMHDs.Width/8*i+k;
5Ø1              b2=b1+1;
5Ø2              if(*b1==*b2)
5Ø3              {
5Ø4                 if(ccount>Ø)
5Ø5                 {
5Ø6                    --ccount;
5Ø7                    write_file(&ccount,1);
5Ø8                    write_file(b1-ccount-1,ccount+1);
5Ø9                    lenbody+=2+ccount;
51Ø                    ccount=Ø;
511                 }
512                 --ccount;
513              }
514              else
515              {
516                 if(ccount<Ø)
517                 {
518                    write_file(&ccount,1);
519                    write_file(b1,1);
52Ø                    ccount=-1;
521                    lenbody+=2;
522                 }
523                 ++ccount;
524              }
525           }
526           if(ccount>=Ø)
527           {
528              write_file(&ccount,1);
529              write_file(b2-ccount,ccount+1);
53Ø              lenbody+=2+ccount;
531           }
532           if(ccount<Ø)
533           {
534              write_file(&ccount,1);
535              write_file(b2,1);
536              lenbody+=2;
537           }
538        }
539     }
```

```
540     if((lenbody%2)!=0)    /* ungerade FileLaenge */
541     {
542        ++lenbody;
543        write_file(&null,1);
544     }
545     if(zeiger!=speicher)
546     {
547        Write(file,speicher,zeiger-speicher);
548        zeiger=speicher;
549     }
550     lenfile=lenheader+lenbody;
551     Seek(file,4,OFFSET_BEGINNING);
552     Write(file,&lenfile,4);
553     Seek(file,lenheader+4,OFFSET_BEGINNING);
554     Write(file,&lenbody,4);
555  }
556  if(zeiger!=speicher) Write(file,speicher,zeiger-speicher);
557  Close(file);
558  if(speicher!=0) FreeMem(speicher,MEM_SIZE);
559  return(TRUE);
560 }
561
562
563 write_file(p,anz)
564 BYTE *p;
565 int anz;
566 {
567  if(MEM_SIZE-(zeiger0-speicher)-anz<0)
568  {
569     Write(file,speicher,zeiger-speicher);
570     zeiger=zeiger0=speicher; /* Ruecksetzen */
571  }
572  while(zeiger-zeiger0<anz)
573  {
574     *zeiger=*p;
575     ++zeiger,++p;
576  }
577  zeiger0=zeiger;
578 }
```

Programm zum Laden eines Bildes:

```
 1 /*********************************************************************/
 2 /* Beispiel zur Verwendung der iff.c-Routinen zum Laden von Bildern */
 3 /*********************************************************************/
 4
 5 #include<exec/types.h>
 6
 7 extern short load();
 8
 9
10 /* Laden von IFF-Bildern mit Angabe des Bild-Files im CLI */
11 main(argc,arg)
12 short argc;
13 char  *arg[];
14 {
15  BYTE status;
16
17  if(argc!=2)
18  {
19     printf("Eingabe: %s [FILE]\n",arg[0]);
20     exit(FALSE);
21  }
22  status=load(arg[1],0);
23  if(status!=0)
24  {
25     printf("Bild kann nicht geladen werden!\n");
26     exit(FALSE);
27  }
28 }
```

Programm zum Speichern eines Bildes:

```
 1 /*********************************************************************/
 2 /* Beispiel zur Verwendung der iff.c-Routinen zum Speichern eines Bildes */
 3 /*********************************************************************/
 4
 5
 6 #include<intuition/intuition.h>
 7 #include<exec/types.h>
 8
 9 #define WIDTH 320
10 #define HEIGHT 250
```

```
#define XSTEP 3
#define YSTEP 3

extern short save();

struct RastPort *rp;          /* Zeiger auf den RastPort        */
struct ViewPort *vp;          /* Zeiger auf den ViewPort        */
struct Screen *MyScreen;      /* Pointer auf die Screen-Struktur */
struct IntuitionBase *IntuitionBase;
struct GfxBase *GfxBase;

/* Nun wird der NewScreen-Struktur definiert */
struct NewScreen MyNewScreen=
 {0,0,WIDTH,HEIGHT,5,15,0,0,CUSTOMSCREEN,0,0,0,0};


short i,x=1,y=1,col=1;



/* Beispiel zum Speichern eines Bildes unter Verwendung der IFF-Routinen */
/* Angabe des Filenamen aus dem CLI                                      */
main(argc,arg)
short argc;
char  *arg[];
{
 if(argc!=2)
 {
    printf("Eingabe: %s [FILE]\n",arg[0]);
    exit(FALSE);
 }
 Open_all();
 Draw_demo();
 /* komprimiertes Speichern des Demo-Bildes */
 if(save(arg[1],1,MyScreen)==FALSE)
    printf("Bild kann nicht gespeichert werden!\n");
 Close_all();
}


Open_all()
{
 if((IntuitionBase=OpenLibrary("intuition.library",0))==0)
    Close_all();
 if((GfxBase=OpenLibrary("graphics.library",0))==0)
    Close_all();
 if((MyScreen=OpenScreen(&MyNewScreen))==0)
    Close_all();
 rp=&(MyScreen->RastPort);
 vp=&(MyScreen->ViewPort);
 Set_colors();
}

Close_all()
{
 if(MyScreen!=0)      CloseScreen(MyScreen);
 if(GfxBase!=0)       CloseLibrary(GfxBase);
 if(IntuitionBase!=0) CloseLibrary(IntuitionBase);
 exit(TRUE);
}

Set_colors()
{
 for(i=0;i<16;++i) SetRGB4(vp,i,i,0,0);
 for(i=0;i<16;++i) SetRGB4(vp,i+16,15,0,i);
}


Draw_demo()
{
 while(MyScreen->TopEdge<100)
 {
    x+=XSTEP; y+=YSTEP;
    if(x>=WIDTH-1) {x=1;++col;}
    if(y>=HEIGHT-1) {y=1;++col;}
    if(col>31) col=1;
    SetAPen(rp,col);
    Move(rp,x,0); Draw(rp,WIDTH-x-1,HEIGHT-1);
    SetAPen(rp,31-col);
    Move(rp,0,y); Draw(rp,WIDTH-1,HEIGHT-y-1);
 }
}
```

Zum Schluß noch eine weitere Anwendung der IFF-Routinen. Nachdem wir in langer Rechenarbeit unsere Bilder berechnet haben, stellt sich das Problem, sie geordnet auf Diskette abzulegen, um sie bei Bedarf ohne große Sucherei finden zu können. Es bietet sich daher an, die Bilder nach Untersuchungsobjekt und Parameter zu ordnen. Um nach all der Arbeit die Bildsequenzen vorführen zu können, habe ich das folgende Programm geschrieben. In einem Spezifikations-File (Spez-File), das mit `ed` erstellt werden kann, werden die Bild-Files angegeben, die nacheinander geladen werden sollen. Dieses Programm wertet diese Datei aus und lädt unter Verwendung der IFF-Routine `load()` die einzelnen Bilder. Ein geladenes Bild bleibt solange bestehen, bis es mit der Maus nach unten gezogen wird. Erst dann wird das nächste File geladen.

Das Spezifikationsfile hat folgenden Aufbau: Beginnt eine Zeile mit ' ' (Space), so wird sie ignoriert. Um also eine Dokumentationszeile einzufügen, beginnt man die Zeile mit einem Leerraum und schreibt dann den gewünschten Text. Auch eine leere Zeile wird ignoriert. Ansonsten wird versucht, ein Bild mit dem in dieser Zeile angegebenen Namen zu laden. Sollte dies nicht gelingen, wird automatisch die nächste Zeile ausgewertet. Ein Beispiel für so eine Datei *(Spez*-File) könnte also wie folgt lauten:

```
 Julia-Bilder
pic0
pic5
 Apfelmännchen
pic10
```

Um dem Programm *present* mitzuteilen, welche Datei als Spezifikationsdatei für das Laden verwendet werden soll, wird beim Starten der Name übergeben, in unserem Fall:

`present.run Spez-File`. Das Programm lädt dann der Reihe nach `pic0`, `pic5` und `pic10`, soweit dies möglich ist.

Programm: *present.c*

```
 1  #include<exec/types.h>
 2  #include<libraries/dos.h>
 3
 4  extern load();
 5
 6
 7  main(argc,argv)
 8  int argc;
 9  char *argv[];
10  {
11   int file;
12   short zeile=0;
13   char line[80];
14
15   if(argc!=2)
16   {
17     printf("Eingabe: %s Spez-File\n",argv[0]);
18     exit(FALSE);
19   }
20   file=Open(argv[1],MODE_OLDFILE);
21   if(file==0)
22   {
23     printf("Spez-File '%s' kann nicht geoeffnet werden!\n",argv[1]);
24     exit(FALSE);
25   }
26   while(inline(line,file,80)!=0)
27   {
28     ++zeile;
29     switch(line[0])
30     {
31       case ' ': /* Rem-Zeile */
32                 break;
33       case 10:  /* Return */
34                 break;
```

```
35        default:  /* sonst Bild-File-Name */
36                  if(load(line,Ø)!=Ø)
37                     printf("Bild %s kann nicht geladen werden!\n",line);
38                  break;
39      }
4Ø   }
41   Close(file);
42 }
43
44
45 inline(c,handle,len) /* eine Zeile des Files einlesen */
46 char c[];
47 long *handle;
48 short len;
49 {
5Ø   short i=Ø,ende=Ø;
51
52   while((i<len)&&(c[i-1]!=1Ø)&&((ende=Read(handle,&c[i],1))!=Ø))
53      ++i;
54   c[i]=Ø;
55   if(i>Ø) c[i-1]=Ø;
56   return(ende);
57 }
```

9 Anhang

Anhang A – Diskette zum Buch

Zuerst einige wichtige Anmerkungen:
Bitte fertigen sie von ihrer Programmdiskette eine Sicherheitskopie an!
Die Programme wurden mit dem Aztec-C-Kompiler entwickelt. Ihre Lauffähigkeit ist aber auch mit dem Lattice C getestet worden.

Auf der beiliegenden Diskette befinden sich alle Programme, die in diesem Buch erscheinen, und einige Bilder im IFF-Format. Die Programme können in der vorliegenden Form nur aus dem CLI gestartet werden. Die Namen dieser stimmen mit den Bezeichnungen aus dem Buch überein, so daß es kein Problem sein dürfte, sie auf der Diskette zu finden.

Die Dateien mit »*.c*« enthalten die Source, die mit »*.run*« sind die dazugehörigen lauffähigen Objekt-Dateien. Die Source-Dateien können mit dem WorkBench-Befehl »`ed ?.c`« editiert werden. Erwartet ein Programm eine Eingabe, so teilt es die Art und Weise mit, in der diese vorgenommen werden soll. Außerdem ist die Bedienung im Buch genau erklärt.

Ein File-Name mit »*.pic*« ist dagegen ein Bild im Iff-Format und kann mit dem Aufruf `Programme/C-Amiga/iff/loaddemo.run ?.pic` geladen werden.

Die Dateien auf der Diskette sind in zwei Directories, Programme und Bilder, getrennt. Das Directory (Schublade) Programme ist in weitere Schubladen unterteilt, die den Kapiteln im Buch entsprechen. Ein Feigenbaum-Programm befindet sich somit in der Schublade Feigenbaum.

Da der Programmumfang des Buches recht erheblich ist, war es nicht mehr möglich, viele Bilder (sogar nur eins!) mit auf der Demo-Diskette unterzubringen. Vielleicht wird sich manch einer daran stören, daß die Programme nicht von der Workbench aus gestartet werden können. Das hat aber seine Gründe: Zuerst einmal erfordern die meisten Programme die Übergabe von Parametern und diese ist nur vom CLI aus, ohne den Umfang der Programme ins endlose zu strecken, möglich. Bei den restlichen Programmen erscheint es mir aber wenig sinnvoll, diese mit Icons zu versehen, denn dadurch würde nur Verwirrung gestiftet. Auch der freie Speicher auf der Diskette begrenzt die Möglichkeiten stark. Ich denke aber, jeder Interessierte wird ohne Probleme die Programme vom CLI aus benutzen können.

Zum Schluß noch eine Anmerkung zur Programmbeendung: Viele meiner Programme können nur mit Herunterziehen des Bildschirms beendet werden. In dieser Methode liegt aber der Vorteil, daß die Abfrage nur eine Programmzeile erfordert, während entsprechende Intuitionverwendungen wesentlich mehr Platz benötigen würden. (Zum Bei-

spiel: Ist nur ein Screen geöffnet, so hilft die schönste Intuition nichts – dazu müßte zuerst ein Fenster geöffnet werden, damit z.B. ein CloseGadget eingerichtet werden kann!)

Im folgenden das Inhaltsverzeichnis der Demo-Diskette (mit »`dir opt a`« ausgegeben):

```
Bilder (dir)
 Mandelbrot-Menge              Mandelbrot-Menge.info
Programme (dir)
    Newton (dir)
      newton.c                    newton.run
      sullivan.c                 sullivan.run
      wurz.c                      wurz.run
    3-D (dir)
          3-d.c                       3-d.run
      3d.dat                      flug.dat
      l.dat                      tetra.dat
      Wuerfel.dat                3D-Generator.c
    Feigenbaum (dir)
      feigenbaum1.c              feigenbaum1.run
      feigenbaum2.c              feigenbaum2.run
      feigenbaum3.c              feigenbaum3.run
      feigenbaum4.c              feigenbaum4.run
      feigenbaum5.c              feigenbaum5.run
      hasen.c                    hasen.run
      pop1                        pop2
      pop3                        pop4
      pop5
    Projektor (dir)
      cos3d2.c                   pro.c
      pro.run                    screen.c
      show
    C-Amiga (dir)
        iff (dir)
          compile.aztec              compile.lattice
          iff.c                      iff.lattice.o
          iff.o                      loaddemo.c
          loaddemo.run                present.c
          present.run                savedemo.c
          savedemo.run                Spez-File
      areafill.c                 areafill.run
      areafill2.c                areafill2.run
      clip.c                      clip.run
      linie.c                    linie.run
      menu.c                        menu.run
      openscreen.c               openscreen.run
      openwindow.c                openwindow.run
      punkt.c                        punkt.run
      rechteck.c                 rechteck.run
Fraktal (dir)
    frame (dir)
      frame.c                          frame.lattice.o
      frame.o
    apfel3d.c                    apfel3d.run
    apfel3d2.c                    apfel3d2.run
    fraktal.c                  fraktal.menu
    fraktal.run                  showinfo.c
    showinfo.run
Ausblicke (dir)
    land (dir)
      land.c                             land.menu
      land.run
    turtle (dir)
      c.c                                c.run
      drachen.inc                        drachen.pic
      drachen.run                        hilbert.inc
      hilbert.run                         koch1.inc
      koch1.run                             koch2.inc
      koch2.run                          tree.inc
      tree.run
    graf.c                       graf.run
    kreis.c                      kreis.run
film (dir)
    film.c                       film.run
    film3d.c                      film3d.run
    filmJulia.c                 filmJulia.run
    filmPlane.c                  filmPlane.run
```

Kurzbeschreibung der Programme

directory Newton

newton.c

Wertet die komplexe Nullstellensuche an Hand der Funktion $z^3-1=0$ aus und zeichnet die Einzugsgbiete der Nullstellen (Attraktoren) `1`, `-.5+i*.866` und `-.5-i*.866`. Aus der Anzahl der notwendigen Iterationen bis zur Erfüllung eines Abbruchkriteriums wir farblich interpretiert.

Aufruf: `newton.run` <n_{max},e_{min},e_{max},r_{min}, r_{max},i_{min},i_{max}><mode>

sullivan.c

Untersucht das Verhalten der Funktion $z^3+(c-1)*z-c$ bezüglich Nullstellensuche mit dem Newton-Verfahren. Auch hier wird die Anzahl der notwendigen Iterationen bis zum Abbruch farblich interpretiert.

Aufruf: `sullivan.run` <n_{max},e_{min},r_{min}, r_{max},i_{min},i_{max}>

wurz.c

Berechnet die komplexen Einheitswurzeln, also die Lösungen der Gleichung $z^n-1=0$ bis zu `n=6`.

Aufruf: `wurz.run`

directory 3-D

3-d.c

Beispielprogramm für dreidimensionale Grafik. Die Körperdefinition wird in einem Text-File *?.dat* vorgenommen. Der Name dieses Files wird dem Programm beim Starten ohne den Zusatz *.dat* übergeben.

Aufruf: `3-d.run ?`

3d.dat, flug.dat, l.dat, tetra.dat und *Wuerfel.dat* sind Demo-Spezifikationsdateien für Körperdefinitionen. Um zum Beispiel *flug.dat* zu verwenden, ist folgender Programmstart notwendig: `3-d.run flug`

3D-Generator.c

Dieses Programm generiert ein *.dat*-File von einer Mandelbrot-Landschaft, das von dem Programm *3-D.c* eingelesen werden kann.

directory Feigenbaum

feigenbaum1.c

Zeichnet ein Feigenbaum-Diagramm. Für jeden auftretenden Wert wird ein Punkt gesetzt, das Diagramm ist also einfarbig. (Formel: $p_{n+1}=p_n*(1+k*(1-p_n))$)

Aufruf: `feigenbaum1.run` <$n_\emptyset$,n_1,k_{min}, k_{max},$p_\emptyset$>

feigenbaum2.c

Wertet analog zu *feigenbaum1.c* die Formel $p_{n+1}=p_n*(1+k*(1-p_n))$ aus, nur die Häufigkeit mit der ein Wert auftritt wird farblich dargestellt.

Aufruf: `feigenbaum2.run` <$n_\emptyset$,n_1,k_{min}, k_{max},$p_\emptyset$>

feigenbaum3.c

Zeichnet ein dreidimenionales Bild des Feigenbaum-Diagramms, sozusagen ein Feigenbaum-Gebirge. Die Höhe des Gebirges entspricht der Häufigkeit des aufgetretenen Wertes. Außerdem wird zugleich diese noch farblich ausgewertet.

Aufruf: `feigenbaum3.run` <$n_\emptyset$,n_1,k_{min}, k_{max},$p_\emptyset$>

feigenbaum4.c

Wertet die Formel $p_{n+1}=p_n*(1+k*(1-p_n))$ nicht mehr wie bisher aus. Bei den Programmen *feigenbaum1.c* und *feigenbaum2.c* wurde nach rechts die Wachstumsrate `k` und nach oben die Werte p_n aufgetragen. In diesem

Programm wird nach rechts p_n und nach oben p_{n+1} aufgetragen.
Aufruf: `feigenbaum4.run` ⟨$n_\emptyset$,n_1,$p_\emptyset$,k_{min}, k_{max},st⟩

feigenbaum5.c

Noch eine andere Auswertemethode wird bei diesem Programm verwendet: Die Folgewerte werden als eine grafische Iteration interpretiert.
Aufruf: `feigenbaum5.run` ⟨n_{max},$p_\emptyset$,k⟩

hasen.c

Berechnet die Vermehrung von Hasen in Abhängigkeit ihres Anfangsbestands, ihrer Wachstumsrate und der Anzahl der betrachteten Durchläufe.
Aufruf: `hasen.run` $p_\emptyset$,k,n
pop1, pop2, pop3, pop4 und *pop5* sind Ausdrucke des Programm *hasen.c* (siehe Buch).

directory Projektor

cos3d2.c

Mit diesem Programm wird eine Filmsequenz aus mehreren Bildern erzeugt. Zur Berechnung des Bildes wird eine cos-Funktion um die y-Achse gedreht. Die Ausgabe erfolgt als schattiertes Flächenmodell unter Berücksichtigung der Beleuchtungsrichtung. Von Bild zu Bild wird eine Phasenverschiebung vorgenommen. Beim Abspielen sieht es dann so aus, als ob eine Welle auseinanderlaufen würde. Wenn die gesamte Phasenverschiebung vom ersten bis zum letzten Bild ein Vielfaches von $2*\pi$ ist, so kann die Sequenz immer wieder hintereinander abgespielt werden, so daß ein fortlaufender Endlos-Film entsteht. In der augenblicklichen Form berechnet das Programm neun Bilder, die insgesamt eine Phasenverschiebung von $2*\pi$ aufweisen. Sie werden als $pic_\emptyset$, pic_1, usw. abgespeichert.
Aufruf: `cos3d2.run`

pro.c

Ist ein Filmprojektor zum Abspielen von IFF-Bildern. Daß die Anzahl der Bilder die eingelesen werden können, durch den Speicher begrenzt ist, leuchtet ein. Da in diesem Programm nur der Chip-Mem, der vom Blitter benutzt werden kann, verwendet wird, beschränkt sich die Anzahl der Bilder bei einem Format von 320 x 200 x 5 auf zehn. Die Abspieldauer etc. kann in einem Steuerfenster nach eigenen Wünschen gewählt werden.
Eine genaue Beschreibung des Programms siehe entsprechendes Kapitel. Die Namen der Bilder, die als Film abgespielt werden sollen, werden beim Aufruf des Programms aus dem CLI mit angegeben. Sie werden dann in der Reihenfolge, in der sie angegeben wurden, geladen. Sollte ein Bild nicht ladbar sein, oder ist zu wenig Speicher frei, meldet das Programm dies und setzt die Ausführung fort.
Aufruf: `pro.run` ⟨$Bild_1$⟩⟨$Bild_2$⟩ ...

screen.c

Öffnet einen Screen ohne Verwendung des Intuition. Diese Initialisierungsroutine ist Ausgangspunkt für den Film-Projektor. Anmerkung: Um die einzelnen Bilder schnell genug anzeigen zu können, muß mit selbst definierten Screens gearbeitet werden. Für jeden, der sich genauer mit dem Programm *pro.c* beschäftigen will, ist es ratsam, erst einmal dieses zu studieren.

show

Ist ein Execute-File, daß das Starten von *pro.run* erleichtern soll. Es hat folgendes Aussehen:

`pro.run` $pic_\emptyset$ pic_1 pic_2 pic_3 pic_4 pic_5 pic_6 pic_7 pic_8 pic_9

Läßt man nun das File mit `execute show` ausführen, versucht *pro.run* die 10 Bilder pic_0,

pic_1, ... zu laden. So erspart man sich bei einer fertig erstellten Filmsequenz vor dem Vorführen immer wieder alle Bildnamen anzugeben.

directory C-Amiga
directory iff

Diese Schublade enthält das IFF-Modul zum Laden und Speichern von IFF-Bildern. Außerdem noch zwei Beispiel-Programme, die die Einbindung verdeutlichen sollen.

iff.c (iff.o / iff.lattice.o)

Ist das IFF-Modul. Sollte ein Programm die Routine `save()` oder `load()` benutzen wollen, so muß das Objekt-Modul *iff.o* dazugelinkt werden. Anmerkung: Das Objekt-Modul erhält man nach dem Kompilieren. *iff.o* ist nur für den Aztec-C-Compiler zu verwenden, *iff.lattice.o* ist für den Lattice C geeignet!

loaddemo.c

Ist ein Ladeprogramm für IFF-Bilder. Es verwendet das IFF-Modul *iff.c*.
Aufruf: `loaddemo.run Bild`

savedemo.c

Ist ein Demo-Programm zum Speichern eines Bildes im IFF-Format. Auch dieses Programm verwendet das IFF-Modul.
Aufruf: `savedemo.run Bild`

present.c

Lädt eine Serie von Bildern. Die Bild-File-Namen werden in einem Text-File angegeben. Der Name dieses Files (zum Beispiel demo) wird beim Starten dem Programm übergeben.
Aufruf: `present.run demo`
Das Text-File hat zum Beispiel folgenden Aufbau:

```
Demo
pic1
pic2
pic3
```

Nach Starten des Programms werden dann die Bilder pic_1, pic_2 und pic_3 geladen. Die Zeilen, an deren ersten Stelle ein Leerzeichen steht, wird ignoriert!

Spez-File

Ist eine Text-Datei der oben beschriebenen Art für das Programm *present.c*

compile.lattice

Execute-File zum Kompilieren mit dem Lattice-C-Compiler, wobei beim Linken das IFF-Modul eingebunden wird.
Aufruf: `ex compile.lattice MyProg`
kompiliert das Programm *MyProg.c*.

compile.aztec

Execute-File zum Kompilieren mit dem Aztec-Compiler unter Einbindung des IFF-Moduls beim Linken.

areafill.c

Ist ein Demo-Programm, da die Initialisierung der Areafill-Routinen verdeutlichen soll.
Aufruf: `areafill.run`

areafill2.c

Läßt ein Dreieck fliegen. Es werden die Initialisierungsroutinen aus dem Programm `areafill.c` verwendet!
Aufruf: `areafill2.run`

clip.c

Ist ein Beispielprogramm für die Verwendung des Blitter mit der `ClipBlit()`-Funktion. In einem Backdrop-Fenster wird ein rechteckiger Bereich um die Maus 16mal mit dem Blitter einkopiert. Bei bestimmten Zustän-

den, das heißt bestimmten Stellungen der Maus, beginnt das Bild zu laufen und wird instabil. Es kann sogar zu einem Rauschen kommen, das durch Bewegungen der Maus gesteuert und manipuliert werden kann. Deshalb ist dieses Programm nicht nur ein Beispiel zur Verwendung des Blitters, sondern zeigt wie seltsam sich Rückkopplungen, die ja die Grundlage für dynamische Systeme sind, auswirken können.
Aufruf: `clip.run`

menu.c

Zeigt, wie ein Menü initialisiert und abgefragt wird.
Aufruf: `menu.run`

openscreen.c

Öffnet einen Screen unter Verwendung des Intuition.
Aufruf: `openscreen.run`

openwindow.c

Öffnet ein Fenster im WorkBench-Screen.
Aufruf: `openwindow.run`

linie.c

Demo-Programm für die `Draw()`-Funktion.
Aufruf: `linie.run`

punkt.c

Ist ein Beispielprogramm für die Verwendung von `WritePixel()`.
Aufruf: `punkt.run`

rechteck.c

Ist ein Beispielprogramm für die Verwendung von `RectFill()`.
Aufruf: `rechteck.run`

directory Fraktal
directory frame

frame.c (frame.o / frame.lattice.o)

Dieses Programm ist wie *iff.c* als Modul gedacht, um in komplexere Programme aufgenommen zu werden. *frame.o* ist wieder nur von Aztec-C-Compiler, *frame.lattice.o* nur für den Lattice C zum Linken verwendbar. Dies Modul übernimmt die Berechnung der Funktionskoordinaten und das Zeichnen der Begrenzungslinien beim Anwählen eines gewünschten Bildbereiches. Es findet beim Programm *fractal.c* Verwendung.

apfel3d.c / apfel3d2.c

Diese beiden Programme zeichnen ein dreidimensionales Bild des Apfelmännchens unter Verwendung der »Rastermethode«. Bei *apfel3d.c* erfolgt die Ausgabe als einfarbiges Flächenmodell. Dagegen wird bei *apfel3d2.c* die Beleuchtung berücksichtigt und das Flächenmodell entsprechend eingefärbt.

fraktal.c

Der Fraktal-Generator! Mit diesem Programm können Apfelmännchen, Julia-Mengen und ihre Verwandten untersucht werden. Sogar Nullstellenuntersuchungen mit dem Newton-Verfahren können ausgeführt werden. Eine Untersuchung eigener Funktionen ist bereits vorgesehen und problemlos durchzuführen. Die Handhabung und Bedienung des Programms ist im Buch genau beschrieben.
Aufruf: `fraktal.run`

fraktal.menu

Ist das Include-File für *fraktal.c*. Es enthält alle wichtigen Definitionen für Menüs etc.

showinfo.c

Ein mit *fractal.c* gespeichertes Bild ist mit einem ».*info*«-File versehen, das alle wichtigen Parameter des Bildes enthält. Um nicht zur Feststellung dieser Parameter immer wieder das gesamte Programm *fractal.run* starten zu müssen ist dieses Programm gedacht. Es liest die ».*info*«-Datei ein und druckt dann die Bildparameter aus. Beim Aufrufen des Programms wird der Bildname angegeben (also ohne ».*info*«!).
Aufruf: `showinfo.run Bild`
(dann wird die Datei *Bild.info* untersucht!)

directory Ausblicke
directory land

land.c

Programm zur Erzeugung fraktaler Landschaften. Sie können sowohl als Gitter- als auch als Flächenmodell ausgegeben und gespeichert werden. Beim Speichern erhält eine Landschaft den File-Namen *land.pic*. Ihre Generierung erfolgt durch zufällige Verschiebung und immer feinere Aufteilung eines Grundmusters.
Aufruf: `land.run`

land.menu

Include-File für das Programm *land.c*.

directory Turtle

Diese Schublade enthält Programme zur Erzeugung »klassischer«, geometrischer Fraktale wie zum Beispiel Koch-Kurven. Wird der Screen in dem gezeichnet wird heruntergezogen, speichern die folgenden Programme das bisher gezeichnete Bild ab. Der Name des Bildes ist dabei im Programm bestimmt. Zum Beispiel speichert *drachen.c* das Bild unter dem Namen *drachen.pic*.

c.c

C-Kurve
Aufruf: `c.run`

drachen.inc

Berechnungsroutine für die Drachen-Kurve. Um das Programm *drachen.run* zu erzeugen kann diese Routine `drachen()` anstatt `c_()` in das Programm *c.c* eingebaut werden. Auch bei den anderen ».*inc*«-Files wird genauso vorgegangen.
Aufruf: `drachen.run`

hilbert.inc

Hilbert-Kurve
Aufruf: `hilbert.run`

koch1.inc

dreieckige Koch-Kurve (Schneekristall)
Aufruf: `koch1.run`

koch2.inc

Rechteckige Koch-Kurve
Aufruf: `koch2.run`

tree.inc

Baum-Kurve
Aufruf: `tree.run`

drachen.pic

Demo-Bild in der Auflösung 640 x 500x2 interlace für einen Drachen-Algorithmus. Um das Bild zu laden geht man folgendermaßen vor:

```
programme/c-amiga/iff/loaddemo.run
drachen.pic
```

graf.c

Beispielprogramm für einfache Rückkopplungen in einer Ebene. Unter der Verwendung der Abbildungsvorschrift

$$y_{n+1} = a - x_n$$
$$x_{n+1} = y_n - \mathrm{sign}(x_n) * \sqrt{(\mathrm{abs}(b*x_n - c))}$$

und dem Startwert (0,0), werden die Folgenglieder auf der Bildebene eingezeichnet. Das Farbregister jedes Punktes, der gezeichnet werden soll, wird erhöht. Dadurch wird die Häufigkeit der auftretenden Punkte sichtbar. Durch die Variation von den Variablen a,b und c und dem Bildausschnitt (rrange, irange) können völlig verschiedene Bilder generiert werden!
Aufruf: graf.run

kreis.c

Mit der Auswertung von (x^2+y^2 modulo 32) wird jedem Bildpunkt eine bestimmt Farbe zugeordnet. (Die Koordinaten x,y entsprechen mit den eigentlichen Bildpunkten!)
Aufruf: kreis.run

directory film

Diese Schublade enthält Programme für die Generierung von fortlaufenden Bildsequenzen. Die Bilder werden im IFF-Format abgelegt und können mit dem »Filmprojektor« *pro.c* abgespielt werden.

film.c / filmPlane.c

Erzeugt eine aus mehreren Bildern bestehende Sequenz der Mandelbrot-Menge. Beim Starten des Programms wird die Anzahl der gewünschten Bilder n, die Rechentiefe n_{max}, und zwei Bereiche angegeben. Wollen wir zum Beispiel 20 Bilder mit einer Rechentiefe von 100 von einem »Zooming« aus Bereich1 (–1,1,–1,1) in Bereich2 (–.1,1,.3,.1) berechnen, so starten wir das Programm wie folgt:

```
film.run 2Ø,1ØØ,-1,1,-1,1,-.1,1,.3,.1
```

Das Programm berechnet dann die 20 Bilder und legt sie im IFF-Format ab (pic_0, pic_1, pic_2, ..., pic_{20}). Das Programm *film.c* berechnet nur einen kleinen Bildausschnitt, während *filmPlane.c* über den gesamten Bildschirm zeichnet.
Aufruf: film.run anz,n_{max},r_{min1},r_{max1}, i_{min1},i_{max1},r_{min2},r_{max2},i_{min2},i_{max2}

filmJulia.c

Entspricht dem Programm *film.c*; es werden jedoch keine Apfelmännchen-Ausschnitte mehr berechnet, sondern Vergrößerungen aus Julia-Mengen. Die Eingabe gewünschter Parameter erfolgt genauso wie bei *film.c* bzw. *filmPlane.c*. Zusätzlich ist hier aber die Angabe der komplexen Konstante $c=_{cre}+_{cim}*i$, die ja für das Aussehen der Julia-Menge maßgegeben ist, möglich.
Aufruf: filmJulia.run anz,n_{max},r_{min1},r_{max1}, i_{min1},i_{max1},r_{min2},r_{max2},i_{min2},i_{max2} cre,cim

film3D.c

Berechnet dreidimensionale Gittermodelle von Apfelmännchen. Auch hier wird eine Bildsequenz berechnet, die als Film abgespielt werden kann. Beim Abspielen der zwei- oder dreidimensionalen Bildfolgen tritt eine unangenehme Eigenschaft der fraktalen Mengen und Grenzen auf, die gerade bei den Gittermodellen negativ auffällt. Der Rand der Mengen ist ja, wie wir mittlerweile wissen, fraktal. Deshalb kann es von einem Punkt zu einem benachbarten zu großen Schwankungen kommen. Dies äußert sich darin, daß im Gittermodell dauernd hohe Spitzen auftauchen und im nachfolgenden Bild wieder verschwinden. Nur die Gesamttendenz bleibt gut zu erkennen.
Aufruf: film3D.run anz,n_{max},r_{min1},r_{max1}, i_{min1},i_{max1}, r_{min2},r_{max2},i_{min2},i_{max2}

directory Bilder

mandelbrot-Menge

Ist ein mit dem Programm *fraktal.c* erzeugtes Bild der Mandelbrot-Menge (quadratische Rückkopplung der Funktion z^2-c über die komplexe Zahlenebene.). Es kann mit ihm wieder geladen und weiterverwendet werden.

mandelbrot-Menge.info

Enthält die notwendigen Bilddaten für das Bild. (Hat nichts mit den »*.info*«-Dateien für die Icons der Workbench zu tun!) Ein Aufruf von `showinfo.run mandelbrot-menge` gibt diese aus:

```
File: mandelbrot-menge.info

Koordinaten: [-1.ØØØØØØ,2.ØØØØØØ][-1.2ØØØØØ,1.2ØØØØØ]
cre: 1.99Ø625  cim:1.18Ø8ØØ
zuletzt berechneter Punkt: [32Ø,249]
nmax: 4Ø  noff: Ø.ØØØØØØ
emin: Ø.ØØØØØØ  emax: 1ØØØØØØØØØØØ.ØØØØØØ
Aufloesung: 32Øx25Ø
Half-Brite-Modus (64 Farben) nicht gewaehlt.
3-D Modus nicht gewaehlt.
Funktion: z2-c (Apfelmaennchen)

Komplexe Dynamische Systeme
```

Info

ist eine Textdatei, die das Inhaltsverzeichnis der Diskette enthält.

Anhang B - Literaturverzeichnis

Bücher zum Thema

1 The Beauty of Fractals – Images of Complex Dynamical Systems
H.–O. Peitgen & P.H. Richter
Springer Verlag, Berlin–Heidelberg–NewYork–Tokyo, 1986

2 Computergrafische Experimente mit Pascal –
Chaos und Ordnung in Dynamischen Systemen
Karl-Heinz Becker & Michael Dörfler
Vieweg, Braunschweig, 1986

3 Die fraktale Geometrie der Natur
B. Mandelbrot
Birkhäuser Verlag, Basel, 1987

4 Spektrum der Wissenschaft –
Sonderheft Computer Kurzweil 1987

5 Das Chaos
Dr. habil. Reinhard Breuer
in Geo: Das neue Bild der Erde / Juli 1985

6 Computer-Graphik – Bilder und Programme zu Fraktalen, Chaos und Selbstähnlichkeit
Uwe Beck
Birkhäuser, 1987
(Programm Praxis – Band 8)

Bücher zum Amiga

7 Amiga –
Systemprogrammierung in C,
John Berry
te-wi Verlag, München

8 Amiga's Programmers Handbook
Mortimore
Sybex-Verlag

9 Amiga Hardware Manual
Commodore

10 Amiga Intuition
Commodore

11 AmigaDos
Commodore

12 ROM Kernel – Library and Devices
Commodore

13 ROM Kernel – Exec
Commodore

14 C auf dem Amiga,
Viertes Amiga-Special
H. Knappe & M. Rahlff
Vogel Verlag, Würzburg

15 Amiga Assembler Buch
Peter Wollschlaeger
Markt&Technik Verlag, München, 1987

16 Amiga Programmierhandbuch
F. Kremser / J. Koch
Markt&Technik Verlag, München, 1987

17 Grafik auf dem Amiga
M. Kohlen
Markt&Technik Verlag, München, 1987

Bücher zur Programmiersprache C

18 Programmieren in C,
Kerningham-Ritchie
Hanser Verlag, München/Wien

19 Der schnelle C-Einstieg
Chip-Special, Vogel Verlag, Würzburg

20 Amiga – C in Beispielen
Dr. E. Huckert
Markt & Technik Verlag, München, 1987

Anhang C - Daten zu den Bildern

Wie schon mehrmals angesprochen, ist es bei ersten Experimenten mit dynamischen Systemen wichtig, Vergleichsergebnisse zu besitzen, um Fehler im eigenen Programm zu finden.

Deshalb sind hier die Parameter der meisten Bilder in diesem Buch festgehalten, so daß sie mit den vorhandenen Programmen leicht selbst erzeugt werden können. Außerdem wird durch ihre Angabe die Suche nach interessanten Ausschnitten erheblich vereinfacht.

Die Bilder sind durch ihre Numerierung leicht zu finden. Hier wird der Typ des Bildes angegeben (zum Beispiel Julia-Menge) und die zur Berechnung wichtigen Werte (zum Beispiel `nmax`, `noff`, `emin`, `emax`, `rmin`, `rmax`, `imin`, `imax` etc.). Außerdem ist das Programm, mit dem die Grafik berechnet wurde, angegeben.

1. Abbildungen

2-1
Feigenbaum-Diagramm
$k_{min}=0$, $k_{max}=3$, $n_0=0$, $n_1=10$, $p_0=.1$
Programm: *feigenbaum1.c*
Aufruf: `feigenbaum1.run Ø,1Ø,Ø,3,.1`

2-2
Feigenbaum-Diagramm
$k_{min}=0$, $k_{max}=3$, $n_0=0$, $n_1=10$ $p_0=.001$
Programm: *feigenbaum1.c*
Aufruf: `feigenbaum1.run Ø,1Ø,Ø,3,.ØØ1`

2-3
Feigenbaum-Diagramm
$k_{min}=1.9$, $k_{max}=3$, $n_0=50$, $n_1=100$, $p_0=.1$
Programm: *feigenbaum1.c*
Aufruf: `feigenbaum1.run 5Ø,1ØØ,1.9,3,.1`

2-4
Feigenbaum-Diagramm
$k_{min}=2.8$, $k_{max}=3$, $n_0=100$, $n_1=500$, $p_0=.1$
Programm: *feigenbaum1.c*
Aufruf: `feigenbaum1.run 1ØØ,5ØØ,2.8,3,.1`

2-5
$n_0=0$, $n_1=1000$, $p_0=.1$, $k_{min}=0$, $k_{max}=3$, $st=50$
Programm: *feigenbaum4.c*
Aufruf: `feigenbaum4.run Ø,1ØØØ,.1,Ø,3,5Ø`

2-6
$n_0=0$, $n_1=500$, $p_0=.1$, $k_{min}=2$, $k_{max}=2.7$, $st=100$
Programm: *feigenbaum4.c*
Aufruf:
`feigenbaum4.run Ø,5ØØ,.1,2,2.7,1ØØ`

2-9
Programm: *feigenbaum5.c*
1) $n_{max}=1000$, p0=.1, k=2.0
   Aufruf: `feigenbaum5.run 1ØØØ,.1,2`
2) $n_{max}=100$, p=.1 und k=2.4
   Aufruf: `feigenbaum5.run 1ØØØ,.1,2`
3) $n_{max}=100$, p=.1 und k=2.4456
   Aufruf: `feigenbaum5.run 1ØØ,.1,2.4456`
4) $n_{max}=100$, p=.1 und k=2.55
   Aufruf: `feigenbaum5.run 1ØØ,.1,2.55`
5) $n_{max}=100$, p=.1 und k=2.8
   Aufruf: `feigenbaum5.run 1ØØ,.1,2.8`

2-10
1) p=k*p*(1-p)
   $n_0=0$, $n_1=100$, $k_{min}=0$, $k_{max}=5$, p0=.1
2) p=k*p*(1-p)
   $n_0=10$, $n_1=50$, $k_{min}=0$, $k_{max}=4$, $p_0=.1$
3) $p=k*p*(1-p^2)$
   $n_0=0$, $n_1=50$, $k_{min}=0$, $k_{max}=3$, $p_0=.1$
4) p=p*sin(p)*cos(p)
   $n_0=10$, $n_1=100$, $k_{min}=0$, $k_{max}=4$, p=.5

2-11
Henon-Attraktor

$x_{n+1}=y_n-a*x_n^2+1$
$y_{n+1}=b*x_n$

1) a=1.4 und b=.3, x(0)=0 und y(0)=0, n=10000
2) n=1000

Die hintereinander auftretenden Folgenwerte wurden miteinander durch eine Gerade verbunden.

4-1-1
Die Mandelbrot-Menge z^2-c
[-1,2.5] [-1.2,1.2]

4-1-2
Geschwindigkeitsdiagramm der Mandelbrot-Menge z^2-c
[-1,2.5] [-1.2,1.2]

4-1-5
Veränderung der Auswertemethode
1) Auswertung mit $im>e_{max}$
   [-1,2.5] [-1.2,1.2]
2) Auswertung mit $im+re>e_{max}$
   [-1,2.5] [-1.2,1.2]

4-1-6

1)	[.4948,.548]	[.6733,.71]
2)	[.8122,.8804]	[.2089,.2558]
3)	[.8304,.8324]	[.2284,.2298]
4)	[.8393,.8406]	[.2239,.2247]
5)	[.8398,.8401]	[.2245,.2247]
6)	[.83991,.83994]	[.22462,.22464]
7)	[.8513,.8687]	[.2284,.2403]
8)	[.8587,.8611]	[.2342,.2358]
9)	[.8592,.8596]	[–.2352,–.2355]
10)	[.9095,.9235]	[.2725,.2821]

4-1-7

1)	[.1388,.1474]	[–1.0220,–1.0161]
2)	[.7433,.7532]	[.1608,.1676]
3)	[.74892,.74895]	[.16489,.16493]
4)	[1.8621,1.8622]	[–.001,.001]
5)	[1.8621,1.8622]	[–.001,.001]
6)	[1.8390,1.9894]	[–.0513,.0513]

4-1-9
Julia-Mengen
1) c=.12+.74*i
2) c=i
3) c=1.25
4) c=–.35+.35*i
5) c=–.5+.3*i
6) c=–.11031–.67037*i

4–1–11
1) c=0 ([–2,2][–2,2])
2) c=.2+.2*i ([–2,2][–2,2])
3) c=.4+.4*i ([–2,2][–2,2])
4) c=.55+.55*i
   1) [–2,2] [–2,2]
   2) [–.6,–.325] [–.432,–.144]
5) c=.8+.8*i
   1) [-2,2] [-2,2]
   2) [-.75,-.4] [-.64,-.16]

4-1-12
»Schnitte« durch die Folge
z^2-c (Im(z)=Im(c)=.5)

4-1-13
$z_{n+1}=z_n*c(1-z_n)$
Untersuchung in der c-Ebene
1) [–1.3945,1.6309] [–1.04,1.04]
2) [.2713,1.4210] [.1872,.9776]
3) [.7479,.8689] [.7176,.8008]
4) [.6227,.6246] [.9012,.9025]

4-1-14
$z_{n+1}=z_n^3+z_n*(c-1)-c$
Untersuchung in der c-Ebene
1) [–.6909,1.3455] [–.67,.73]
2) [–.2359,.3750] [.0160,.4360]

4-1-20
Bewertungskriterium: $y>e_{max}$
1) c=1.25
2) c=-i

4-2-1
$e_{min}=.001$, $e_{max}=100000$, $n_{max}=100$
[–1.5,2.5] [–1.2,1.2]

4-2-2
$e_{min}=.001$, $e_{max}=10000$, $n_{max}=100$
[.7453444,.745896] [.112954,.1130534]

4-2-3
$e_{min}=.001$, $e_{max}=100000$, $n_{max}=150$
[.745054,.745538] [.112905,.113236]

4-2-4
$e_{min}=.001$, $e_{max}=100000$, $n_{max}=150$
[.74448,.74657] [.11196,.11339]

4-2-5
$e_{min}=.001$, $e_{max}=100000$, $n_{max}=150$
[.7408,.75104] [.10835,.11536]

4-2-6
$e_{min}=.001$, $e_{max}=100000$, $n_{max}=150$
[1.044,1.172] [–.2992,–.2116]

4-2-7
$e_{min}=.001$, $e_{max}=100000$, $n_{max}=150$
[.9,.92] [–.255,–.275]

4-2-8
$e_{min}=.001$, $e_{max}=100000$, $n_{max}=150$
[.857,.867] [–.27,–.26]

4-3-11
$z^3+(c-1)*z-c-1=0$
(1) [–2,2] [–2,2]
(2) [–.2,.4] [1.5,1.9108]

4-4-3-1
$\text{int}(x^2+y^2) \text{ modulo}(2)$
[–20,60] [–20,60]

4-4-3-2
$\text{int}(x^3+y^3) \text{ modulo}(2)$
[–10,10] [–10,10]

4-4-3-3
$\text{int}(x^4+y^4) \text{ modulo}(2)$
[–1,10] [–10,10]

4-4-3-4
$\text{int}(1000*(\cos(x)+\sin(x))) \text{ modulo}(2)$
[–2,2] [–2,2]

4-4-3-5
$x_{n+1}=y_n-\text{sgn}(x_n)*\sqrt{(\text{abs}(b*x_n-c))}$
$y_{n+1}=a-x_n$
Anfangswerte x0=x0=0, a=.4, b=1 und c=0
[–2.5,2.5] [–2.5,2.5]

4-4-3-7
$x_{n+1}=y_n-\text{sgn}(x_n)*\sqrt{(\text{abs}(b*x_n-c))}$
$y_{n+1}=a-x_n$
Anfangswerte x0=x0=0, a=–3.14 und b=c=.3

4-4-3-8
$x_{n+1}=y_n-\sin(x_n)$
$y_{n+1}=a-x_n$
mit a=3.1

4-4-3-9
$x_{n+1}=y_n-\sqrt{(x_n^2-.2)}$
$y_{n+1}=a-x_n$
mit a=3.1

4-4-3-10
$x_{n+1}=y_n-\text{sgn}(x_n)*\sqrt{(\text{abs}(b*x_n-c))}$
$y_{n+1}=a-x_n$
a=.1, b=1 und c=0
a=.4, b=1 und c=0
a=.7, b=1 und c=0
a=1.4, b=1 und c=0
a=4, b=1 und c=0

4-4-3-11
$$x_{n+1} = y_n + a * x_n + \frac{2 * (1-a) * x_n^2}{(1+x_n^2)}$$

$$y_{n+1} = -x_n + a * x_{n+1} + \frac{2 * (1-a) * x_{n+1}^2}{(1+x_{n+1}^2)}$$

1) a = –.6 / Startwerte r0 = i0 = 1
2) a = –.4 / Startwerte r0 = i0 = 1
3) a = –.3 / Startwerte r0 = i0 = 1
   a = –.2 / Startwerte r0 = i0 = 1
   a = .1 / Startwerte r0 = i0 = 1
   a = .3 / Startwerte r0 = i0 = 1
4) a = .3 / Startwerte r0 = i0 = 30
5) a = .3 / Startwerte r0 = i0 = 26
6) a = .3 / Startwerte r0 = i0 = 7
   a = .3 / Startwerte r0 = i0 = 5
   a = .3 / Startwerte r0 = i0 = 2
   a = .3 / Startwerte r0 = i0 = 1.5

2. Bildteil

Tafel 1
Feigenbaum-Diagramm.
$k_{min} = 2.0$, $k_{max} = 3.0$, $n_0 = 0$, $n_1 = 100$, $p_0 = .1$
Programm: *feigenbaum2.c*
Aufruf: `feigenbaum2.run 0,100,2,3,.1`

Tafel 2
Dreidimensionales Feigenbaum-Gebirge.
$k_{min} = 2.0$, $k_{max} = 3.0$, $n_0 = 0$, $n_1 = 100$, $p_0 = .1$
Programm: *feigenbaum3.c*
Aufruf: `feigenbaum3.run 0,100,2,3,.1`

Tafel 3
Verschiedene Stadien einer laufenden Cosinus-Welle.
Programm: *cos3d2.c*
Aufruf: `cos3d2.run`

Tafel 4
Tapetenmuster.
`int(x²+y²) modulo(32)`

Tafel 5
Tapetenmuster.
`int(x³+y³) modulo(32)`
(–10,10) (–10,10)

Tafel 6
Tapetenmuster.
`int(x⁴+y⁴) modulo(32)`
(–10,10) (–10,10)

Tafel 7
Tapetenmuster.
`int(1ØØØ*(cos(x)+sin(y))`
(–10,10) (–10,10)

Tafel 8 ((1), (2), (3) und (4))
Ein Flug ins Apfelmännchen (`z²-c`).
Zoom aus Bereich [.74448,.74657] [.11196,.11339] (erstes Bild) in [.745054,.745538][.112905,.113236] (letztes Bild).

Tafel 9
Rückkopplungen in der komplexen Zahlenebene.
$x_{n+1} = y_n - sgn(x_n) * \sqrt{(abs(b * x_n - c))}$
$y_{n+1} = a - x_n$
a = .4, b = 1, c = 0, [–2.5,2.5][–2.5,2.5]

Tafel 10
Eine künstliche Landschaft aus dem Computer.

Tafel 11 ((1), (2) und (3))
»Verzerrungen« – antiparallele Schnitte durch die Funktion z^2-c
(`Im(z)=Im(c)=.5` wurden festgehalten / `Re(z)` und `Im(c)` werden durch den augenblicklichen Bildpunkt bestimmt.)
Die Bilder (2) und (3) sind jeweils Vergrößerungen aus (1).

Tafel 12
Auswertung der komplexen, rekursiven Folge $z^3+(c-1)*z-c$ nach dem Attraktor 1 in der komplexen c-Ebene mit Hilfe des Newton-Verfahrens (nach Sullivan, Garnett und Curry).
[-.2,.4] [1.5,1.9108]

Tafel 13
Auswertung der Nullstellensuche mit Hilfe des Newton-Verfahrens an der Funktion z^3-1. Die Einzugsgebiete der drei Nullstellen sind grün, rot und blau eingefärbt. Besonders interessant sind die Grenzstrukturen zwischen den Attraktionsgebieten.

Tafel 14 ((1), (2) und (3))
Geschwindigkeitsdiagramme der Untersuchung von $z^3-1=0$ mit Hilfe des Newton-Verfahrens.
1) Nullstelle -.5-.886*i
2) Nullstelle -.5+.886*i
3) Nullstelle 1

Tafel 15 ((1), (2) und (3))
Geschwindigkeitsdiagramme von Vergrößerungen der komplexen Konvergenzebene. Es wurde die Konvergenz gegen alle drei Nullstellen ausgewertet.

Tafel 16
Auswertungen mit dem Newton-Verfahren.
1) $z^4-1=0$ (alle Nullstellen)
2) $z^5-1=0$ (gegen alle Nullstellen)
3) $z^5-1=0$ (gegen eine Nullstelle)
4) $z^5-1=0$ (gegen alle Nullstellen)
5) $z^5-1=0$ (Ausschnitt aus Bild (4))

Tafel 17
Das Apfelmännchen.
(Auswertung der Folge z^2-c in der komplexen c-Ebene).
[-1,2.5] [-1.2,1.2]
1) Abbruchkriterium: $re^2+im^2>e_{max}^2$
2) Abbruchkriterium: $im^2>e_{max}^2$

Tafel 18
Die Folge z^3-c / Auswertung in der c-Ebene.
1) Die gesamte Menge
2) (Half-Brite-Modus – 64 Farben)
   Vergrößerung einer Spitze, die in die gesamte Menge ragt
3) Vergrößerung einer Spirale aus Bild (2)
4) Vergrößerung der großen Ausbuchtung an der linken Seite der gesamten Menge aus Bild (1)

Tafel 19
Dreidimensionale Mandelbrot-Landschaften.
1) das gesamte Apfelmännchen
2) Vergrößerung aus dem Randbereich

Tafel 20
Auschnitte am Rand der Mandelbrot-Menge
1) (0.1047,0.1907) (–1.0096,–1.0687)
2) (0.1432,0.1435) (–1.0188,–1.0190)
Auf der Spitze der Mandelbrot-Menge
3) (1.6264,1.9013) (–0.1,0.1)
4) (1.8586,1.8641) (–0.0013,00025)
An diesen vier Bildern erkennt man eindrucksvoll das Phänomen der Selbstähnlichkeit. Bei sehr großen Vergrößerungen findet man kleine Apfelmännchen, die ihrem großen Vorbild in nichts nachstehen.

Tafel 21
Weitere Vergrößerungen des Apfelmännchenrandes.
1) (1.8607,1.8610) (0.0006,0.0008)
2) (0.4080,0.7170) (0.5125,0.7250)
3) (0.5206,0.6288) (0.6984,0.7069)
4) (0.4621,0.6288) (–0.7073,–0.5927)

Tafel 22
Julia-Mengen (z^2-c)
(64 Farben / Extra-Half-Brite-Modus).
1) $n_{max}=500$, $e_{min}=0.000001$, $e_{max}=100000$
   $c=0.110310-0.67037*i$
   (–1.5,1.5) (–1.5,1.5)
2) $n_{max}=100$, $e_{min}=0.001$, $e_{max}=1000$
   $c=0+i$
   (-1.5,1.5) (-1.5,1.5)
3) $n_{max}=100$, $e_{min}=0.001$, $e_{max}=1000$
   $c=1.25$
   (-1.5,1.5) (-1.5,1.5)

Tafel 23
Julia-Mengen (z^2-c)
(64 Farben / Extra-Half-Brite-Modus).

Tafel 24
Julia-Mengen in räumlicher Darstellung (z^2-c)
(32 Farben / 5 Bit-Planes).
1) $c=-0.32-0.043*i$
(-1.5,1.5) (-1.5,1.5)
2) $c=-0.11031-0.67037*i$
(-1.5,1.5) (-1.5,1.5)

Tafel 25
Julia-Mengen (z^2-c)
(64 Farben / Extra-Half-Brite-Modus).

Tafel 26
Julia-Mengen.
(64 Farbe / Extra-Half-Brite-Modus).

Tafel 27
Julia-Mengen
(64 Farben / Extra-Half-Brite-Modus).

Tafel 28
Julia-Mengen »höherer« Ordnung
(64 Farben / Extra-Half-Brite-Modus).
1) z^3-c
$n_{max}=100$, $e_{min}=0.001$, $e_{max}=1000$
$c=0.6+0.6*i$
(-1.2,1.2) (-1.2,1.2)
2) z^4-c
$n_{max}=100$, $e_{min}=0.001$, $e_{max}=1000$
$c=0.5+0.5*i$
(-1.5,1.5) (-1.5,1.5)
3) Vergrößerung aus (2)
$n_{max}=100$, $e_{min}=0.001$, $e_{max}=1000$
$c=0.5+0.5*i$
(-0.6375,-0.525) (-0.396,-0.252)
4) z^5-c
$n_{max}=100$, $e_{min}=0.001$, $e_{max}=1000$
$c=0.77+0.77*i$
(-1.2,1.2) (-1.2,1.2)

Anhang D - Sachworterklärungen

Ableitung

y=f(x) sei eine reelle Funktion. Dann beschreibt ihre erste Ableitung f'(x) wie stark sich y in Abhängigkeit von x ändert. Sie gibt also die Steigung der Kurve von f(x) an. Die zweite Ableitung f''(x) macht eine Aussage über die Krümmung der Funktion. Ableitungen werden verwendet, um Extremstellen oder Wendestellen zu finden und zu charakterisieren.

Absolutwert

Der Betrag (=Absolutwert) einer reellen Zahl z ist recht einfach zu berechnen. Er ist der »positive« Teil der Zahl. D.h., der Betrag von –5 ist zum Beispiel 5, der Betrag von 8 ist gleich 8. Interessanter ist der Betrag einer komplexen Zahl: Hier ist der Betrag der Zahl z=re+i* im gleich ihrem Abstand vom Ursprung. Nach dem Satz des Pythagoras: Betrag von z ist gleich

$\sqrt{(re*re+im*im)}$

(z.B. Betrag von z=3+4*i ist gleich

$\sqrt{(3*3+4*4)}=5$).

Siehe auch Argument bzw. Phase komplexer Zahlen.)

Algorithmus

Ein Algorithmus ist eine Arbeitsanleitung, die Schritt für Schritt so formuliert ist, daß sie von einem Rechner (oder durchaus z.B. auch einem Menschen) eindeutig abgearbeitet werden kann.

Apfelmännchen

Ein Apfelmännchen ist ein anderer Name für die Mandelbrot-Menge. Dieser »Spitzname« wurde von der Bremer Forschergruppe für die von B. Mandelbrot 1980 entdeckte Menge vergeben. (Sieht die Menge wirklich wie das Innere eines Apfels aus? – Das bleibt wohl der Phantasie überlassen.)

Argument

Ein Argument (oder auch Phase) einer komplexen Zahl gibt den Winkel, den die Verbindungsgerade Ursprung – komplexe Zahl (in der komplexen Zahlenebene als Punkt dargestellt!) mit der x-Achse einschließt, gegen den Uhrzeigersinn an. Betrag und Phase beschreiben eine komplexe Zahl ebenso wie Real- und Imaginärteil. (Zur Umrechnung der beiden Darstellungen siehe Kapitel 3 – Mathematische Grundlagen.)

Attraktionsgebiet

Wird eine rekursive Folge mit Werten aus dem Attraktionsgebiet bzw. Einzugsgebiet gestartet, so »landet« sie unweigerlich bei dem zu diesen Gebieten gehörigen Attraktor.

Attraktor

Eine konvergente, rekursive Folge strebt gegen einen Grenzwert. Dieser Grenzwert wird auch als Attraktor bezeichnet, da er die Folgenwerte gleichsam an sich »heranzieht«. Bei diesem Attraktor muß es sich nicht um einen Wert handeln. Wie bei dem Feigenbaum-Diagramm kann der Attraktor zu einer ganzen Figur (in diesem Fall dem Feigenbaum) ausarten.

Betrag

Siehe Absolutwert.

Chaos

Ist der Gegensatz der Ordnung und Berechenbarkeit. Für unsere Betrachtung: Der ordentliche Zustand bedeutet Berechen-

barkeit. Auf der anderen Seite steht das Chaos – die Unvorhersagbarkeit der Vorgänge.

Chaos-Forschung
Ist ein neuer Forschungszweig, in dem der »Übergang zwischen Ordnung und Chaos« untersucht wird. Die Forscher sind zumeist Mathematiker oder Physiker (oder auch Computergrafik-Freaks).

Chaos-Theorie
Diese Theorie zeigt, entgegen früheren Annahmen, daß die »Berechenbarkeit der Welt« prinzipiell nicht möglich ist. Für manche Systeme ist es einfach nicht möglich vorauszusagen, wie diese sich verhalten werden. (Diese Erkenntnis beschränkt sich nicht nur auf technische und naturwissenschaftliche Phänomene, sondern trifft durchaus auch auf wirtschaftliche und politische Vorgänge zu!)

Delta (d), Naturkonstante
Diese Konstante beschreibt den Zusammenhang der einzelnen Teilungspunkte beim Feigenbaum-Diagramm, findet sich aber durchaus auch bei ganz anderen Phänomenen. Deshalb spricht man auch von einer Naturkonstante. Sie wurde von M. Feigenbaum entdeckt und wird durch die folgende Formel beschrieben:

$$d = \frac{(k_n - k_{n-1})}{(k_{n+1} - k_n)}$$

mit n gegen Unendlich.

Differentialgleichung
Eine Differentialgleichung enthält eine Funktion f(x) und eine oder mehrere Ableitungen dieser Funktion. Zum Beispiel:
$f(x)+f'(x)=x$

Dimension, fraktale
Einem Fraktal kann eine Dimension zugeordnet werden. Diese ist (meist) gebrochen rational. Es handelt sich zum Beispiel um ein fraktales Gebilde zwischen Linie und Fläche. (Die Definition des fraktalen kann über solche gebrochenen Dimensionen erfolgen.)

Divergenz
Strebt eine Folge gegen unendlich, so wird diese als (bestimmt) divergent bezeichnet.

Doppelpendel
Eine kreisförmige drehbar gelagerte Scheibe, auf deren Rand wiederum eine weitere Scheibe drehbar gelagert ist, wird als Doppelpendel bezeichnet.

Einzugsgebiet
Siehe Attraktionsgebiet.

Euler-Verfahren
Ist ein Näherungsverfahren zur Lösung von Differentialgleichungen.

Fatou, Pierre
Französischer Mathematiker, der sich bereits vor 1920 (siehe G. Julia) mit dem Verhalten komplexer Iterationen beschäftigte.

Feigenbaum, Mitchell
Der Physiker Mitchell Feigenbaum entdeckte den Zusammenhang der Naturkonstante »Delta«. Die Feigenbaum-Diagramme bzw. Feigenbäume tragen deshalb seinen Namen.

Fixpunkt
Für einen Fixpunkt einer Funktion f(c) gilt: $f(c)=c$ (c komplexe Zahl). Die Funktion bildet sich also für diesen Wert c auf sich selbst ab – deshalb Fixpunkt.

Folge

Eine Folge ist eine eindeutige Abbildungsvorschrift, die den Elementen einer Menge neue zuordnet. Die abgebildeten Elemente werden als Folgenglieder bezeichnet.

Folge, zyklische

Wechseln sich die Werte einer Folge immer in der gleichen Reihenfolge hintereinander ab, so wird die Folge als zyklisch bezeichnet. Die Anzahl der Werte, die sich wiederholen, geben die Ordnung an (z.B.: 2,3,5,2,3,5,2,3,5... ist eine zyklische Folge dritter Ordnung).

Folge, rekursive

Eine rekursive Folge ist die Grundlage für jede Art von Fraktal, denn deren wichtigstes Merkmal ist die Rückkopplung! Bei einer rekursiven Folge wird das Ergebnis einer Abbildungsvorschrift wieder als Startwert für die Berechnung des nächsten Folgenglieds verwendet.

Fraktal

Ein Gebilde, dessen Dimension gebrochen rational ist (Ausnahmen bestätigen die Regel), wird als Fraktal bezeichnet. Ein Fraktal kann zum Beispiel nicht eindeutig einer Linie (endliche Länge!) bzw. einer Fläche zugeordnet werden.

Geschwindigkeitsdiagramme

Siehe Höhenlinien.

Grenze

Eine fraktale Grenze trennt zwei oder mehrere Attraktions-(Einzugs-)gebiete voneinander ab.

Grenzwert

Eine konvergente Folge strebt gegen einen festen Wert. Dieser wird als Grenzwert bezeichnet.

Hidden-Lines

Sind verdeckte Linien bei dreidimensionalen Objektdarstellungen. Werden sie unterdrückt, so wird aus einem Gitter- ein Flächenmodell.

Höhenlinien

Entstehen bei der Auswertung der benötigten Folgendurchläufe n bis zum Zutreffen einer Abbruchbedingung. Sie sind auf die Modulo-Funktion zurückzuführen. Die Höhenlinien bilden die Möglichkeit zu beurteilen, wie schnell die komplexe rekursive Folge konvergiert bzw. divergiert. (Deshalb werden die Bilder, die bei dieser Auswertung entstehen, auch Geschwindigkeitsdiagramme genannt.)

Iteration

(Wiederholung, schrittweise Annäherung). Bezeichnet ein mathematisches Verfahren, dessen unbegrenzte Wiederholung die Möglichkeit gibt, eine Näherungslösung der exakten Lösung beliebig genau anzunähern.

Imaginärteil

(mathematisches Zeichen Im()). Der Imaginärteil ist der in einer komplexen Zahl $z=x+i*y$ bei $i=\sqrt{(-1)}$ stehende Teil, also $Im(z)=y$.

Julia, G.

Französischer Mathematiker, der sich bereits vor 1920 mit komplexen rekursiven Folgen beschäftigte. Ihm zu Ehren werden alle fraktalen Gebilde, die auf Grund komplexer Folgen entstehen, auch Julia-Mengen genannt.

Julia-Mengen

Entstehen bei der Auswertung der komplexen rekursiven Folge

$$z_{(n+1)} = z_n^2 + c.$$

Kaskadenverzweigung

Als (periodenverdoppelnde) Kaskadenverzweigung wird das Teilen (bei jeder Teilung verdoppelt sich die Anzahl der Feigenbaum-»Zweige«) des Feigenbaums bezeichnet.

Kausalitätsprinzip

Das Kausalitätsprinzip sagt aus, daß ähnliche Ursachen auch ähnlich eindeutige und vorhersagbare Wirkungen haben. Die Chaostheorie weist die Grenzen dieses Prinzips auf.

Konjugiert Komplexe

Die konjugiert Komplexe z* einer komplexen Zahl z=re+i*im wird gebildet, indem das Vorzeichen des Imaginärteils umgekehrt wird, also z* =re-i*im. (z.B.: z=3-4*i dann ist: z* =3+4*i)

Konvergenz

(das Gegenteil von Divergenz). Strebt eine Folge gegen einen festen Wert für (unendlich) viele Durchläufe, so wird dieser als Grenzwert bezeichnet und die Folge konvergent genannt.

Koordinaten, karthesische

Eine Ebene wird durch zwei aufeinander senkrecht stehende Koordinatenachsen aufgespannt. Diese Ebene wird als karthesisches Koordinatensystem bezeichnet. Es eignet sich zum Beispiel komplexe Zahlen grafisch darzustellen. Dazu wird der Imaginärteil auf der einen Achse und der Realteil auf der anderen dargestellt. Der so zugeordnete Punkt entspricht eindeutig der komplexen Zahl. In diesem Koordinatensystem kann die Addition, bzw. Subtraktion zweier komplexer Zahlen einfach verdeutlicht werden (Vektoraddition).

Koordinaten, polare

Im Vergleich zu den karthesischen Koordinaten wird die Lage eines Punktes nicht mehr durch seine Projektionen auf die Achsen, sondern durch seinen Abstand vom Ursprung und dem Drehwinkel beschrieben. Auch die Polarkoordinatendarstellung von komplexen Zahlen ist von großer Bedeutung. Der Betrag und die Phase (Argument) der komplexen Zahl ordnet den Punkt in der Zahlenebene zu. Diese Darstellung macht eine eine Verdeutlichung der Multiplikation von komplexen Zahlen möglich (Multiplikation zweier komplexen Zahlen bedeutet Multiplikation ihrer Beträge und Addition ihrer Argumente)

Koordinaten, relative

(auch homogene Koordinaten oder Verhältniskoordinaten genannt). Soll ein kathesisches Koordinatensystem auf dem Bildschirm dargestellt werden, so muß jedem Bildpunkt eindeutig genau ein Wert zugeordnet werden. Dazu ist eine Umrechnung (auch Transformation genannt) der Pixelkoordinate des Bildschirms in einen Punkt der Zahlenebene notwendig. Dies geschieht einfach mit Hilfe des Dreisatzes.

Mandelbrot, B.

Ist ein aus Polen stammender Mathematiker (lebt heute in den USA), der eine sehr bedeutende Rolle in der Erforschung des Übergangs zwischen Chaos und Ordnung spielt. Er hat den Begriff des Fraktalen definiert und geprägt.

Mandelbrot-Menge

(siehe auch Apfelmännchen). Diese legendäre Menge (rekursive Komplexe Folge $z_{n+1}=z_n^2+c$) wurde von B. Mandelbrot 1980 entdeckt und trägt deshalb seinen Namen.

Mathematik, experimentelle
Ist ein neuer Forschungszweig der Mathematik, der mit dem Versuchslabor »Computer«, den Übergang zwischen Chaos und Ordnung untersucht (siehe auch Chaosforschung).

Matrix
Eine Matrix ist ein rechteckiges Zahlenschema, dessen Zahlen als Elemente bezeichnet werden. Das Zahlenschema unterteilt sich in Spalten und Zeilen. Die Anzahl der Spalten n und der Zeilen m bestimmen die Ordnung m*n der Matrix. Die Bedeutung der Matrix liegt in der Anwendbarkeit zur Lösung von linearen (auch DGL-)Gleichungssystemen.

Matrixmultiplikation
Beschreibt die Multiplikation zweier Matrizen.

Newton-Verfahren
Ist ein Nährungsverfahren zur Berechnung von Nullstellen einer reellen oder auch komplexen Funktion. Nach unendlich vielen Iterationen wird die gesuchte Nullstelle beliebig genau angenähert (soweit der Anfangsnäherungswert richtig gewählt wurde).

Ordnung im Chaos
Obwohl unter Chaos meist Unordnung verstanden wird, zeigt sich bei der Untersuchung komplexer dynamischer Systeme, daß mathematisches Chaos ein hohes Maß an Ordnung aufweist. Man denke hier nur an die periodenverdoppelnde Kaskadenverzweigung der Feigenbäume oder die Selbstähnlichkeit beim Apfelmännchen.

Periodenverdopplung
Siehe Kaskadenverzweigung.

Phase
Siehe Argument.

Polarkoordinaten
Siehe polare Koordinaten.

Rückkopplung
(Siehe auch rekursive Folgen.) Rückkopplung ist ein entscheidendes Element von fraktalen Systemen. Unter dieser versteht man die ständige Beeinflussung des folgenden Wertes durch den vorherigen. Im mathematischen Sinne bedeutet dies, daß ein berechneter Wert als Grundlage für die Berechnung des nächsten herangezogen wird. Alle realen physikalischen Vorgänge in der Natur unterliegen dieser Rückkopplung!

Realteil
(Mathematische Schreibweise Re().) Der Realteil einer komplexen Zahl $z=x+i*y$ ist der Teil, der nicht bei $i=\sqrt{(-1)}$ steht, also $Re(z)=x$ (siehe auch Imaginärteil.)

Selbstähnlichkeit
Ist ein sehr wichtiges Phänomen von fraktalen Grafiken. Es beschreibt die Eigenschaft, daß bei beliebigen Vergrößerungen immer wieder »ähnliche« Strukturen gefunden werden, die der Ausgangsfigur sehr stark gleichen.

Struktur des Chaos
Siehe Ordnung des Chaos.

Skalierung
Soll ein dreidimensionaler Körper seine Größe ändern, so muß dieser neu skaliert werden. (Die x-, y- und z-Koordinate werden mit Konstanten multipliziert und bestimmen so die neue Größe des Körpers.)

sqrt(x)
Bedeutet die zweite Wurzel aus X. (Sqrt steht für square root – Quadratwurzel.)

Szene
Werden einzelne Punkte zu Kanten, Flächen und verschiedenen Körpern zusammengefaßt, so entsteht eine dreidimensionale Szene.

Teilungspunkt
Die Punkte, an denen sich der Feigenbaum teilt (siehe periodenverdoppelnde Kaskadenverzweigung), nennt man Teilungspunkte.

Transformation
Werden in der Mathematik den Elementen einer Menge eindeutig neue zugeordnet so spricht man von Transformationen. Bei der dreidimensionalen Grafik bedeutet Transformation Größenveränderung (Skalierung), Verschiebung, Drehung (oder Ähnliches) eines Körpers.

Turtle-Grafik
Diese Art der Grafik ist Grundlage für die Programmiersprache LOGO. Man geht von einer kleinen Schildkröte auf dem Bildschirm aus, die über diesen bewegt werden kann. Dies geschieht, indem man sich in sie »hineinsetzt«. Es kann dann eine bestimmte Strecke vorwärts oder rückwärts gefahren oder die Laufrichtung um einen bestimmten Winkel gedreht werden. Dadurch, daß ein Bewegungsablauf immer von dem vorherigen Standpunkt und der Laufrichtung abhängt (Rekursion, Rückkopplung), eignet sich diese Grafik besonders zur Generierung fraktaler Grafiken.

Vektor
Ist in der geometrischen Deutung eine mit einem bestimmten Richtungssinn versehene Strecke, die durch einen Pfeil verdeutlicht wird. Dieser Pfeil kann beliebig parallel verschoben werden. Unter Betrag des Vektor versteht man die Länge des Pfeils und unter seinem Argument die Richtung (siehe Polarkoodinatendarstellung).

Verhulst
Formulierte bereits im Jahre 1845 eine mathematische Gleichung, die das Wachstum von Tieren beschreibt. Diese Gleichung ist Grundlage für die Feigenbäume.

Verkrumpelung
Ist ein wesentliches Merkmal von fraktalen Grafiken. Man versteht darunter die Eigenschaft, daß fraktale Grenzen unendlich lang und verzweigt sind – eben verkrumpelt.

Wurzel, komplexe
Auch aus komplexen Zahlen lassen sich Wurzeln »ziehen«. Diese sind allgemein wieder komplexe Zahlen. Zur Berechnung geht man am besten von der Polarkoordinatendarstellung von komplexen Zahlen aus (siehe Kapitel 3 – Mathematische Grundlagen).

Zahl, imaginäre
Siehe Imaginärteil.

Zahl, reelle
Siehe Realteil.

Zahlenebene (Gaussche, komplexe)
Siehe karthesische bzw. polare Koordinaten.

Zahlenfolge
Siehe Folge.

Stichwortverzeichnis

A

Abbildungsfunktion 44
Abbruchkriterium 44, 47
Abbildungsvorschrift 41
Abfrage 207
Abhängigkeit 26
Ableitung 267
Absolutwert 267
Abspielgeschwindigkeit 133
Achsensymmetrie 52, 60
Addition 39
Additionstheoreme 40
Adress-Operator 20
Alerts 213
Algorithmus 51, 267
Amiga 7, 13, 97
Amiga-DOS 228
Amiga-Taste 150
Anfangsparameter 61
Anfangswert 28
Animation 76, 190, 235
apfel3d.c 74
Apfelmänchen 8, 13
–, Körper 50
Arbeitsfenster 151
Arcus-Cosinus 182
areafill.c 222, 253
AreaInfo-Struktur 222
Argument 39, 267
Array 100
ASCII-Code 215
ASCII-String 220
Assembler 204
Attraktionsgebiete 86
Attraktor 29, 42
Aufgabenstellung 17
Auflösung 55, 149
Auflösung, hohe 21
–, niedrige 201
Aufwärtskompatibilität 233
Ausgabekanal 207
Ausgabewerte 17
Auswertung 16
–, quantitativ 21
–, zahlenmäßig 29
Auswertungskriterien 26
Auswertungsmethode 32
Aztec C 147, 237

B

Backdrop-Fenster 224, 253
Baum-Kurve 113
Beleuchtungsvektor 71, 101
Berechnungsroutine 255
Bestimmungsgleichung 37, 51
Betrag 267
Betriebssystem 201
Bewegung, chaotische 11
Bild-File 135
Bildaufbau 218
Bildbegrenzung 148
Bilddaten 133, 257
Bilder, zweidimensionale 71
Bildfolge 57
Bildformat 148
Bildpunkt 55, 108, 256
–, Koordinaten 151
Bildschirm 173
Bildschirmebene 71
Bildschirminhalt 219
Bildschirmrasterung 71
Bildschirmzeile 186
Bildvektor 174
Biologie 11
BitMap-Struktur 134, 204
Bitplane 202
Blitter 133, 187
BMHD-Chunk 234
BODY-Chunk 234

C

C-Kurve 113
CAMG-Chunk 235
Chaos 15, 20, 87, 267
–, Forschung 10, 12, 32, 268
–, Theorie 268
Chaosbereich 31
CheckMark 209
CHIP-Memory 133
Chaotisches Verhalten 11
Chunk 233
CLI 18, 22, 249
–, Fenster 148
clip.c 253
clip2.c 224
Clipping-Algorithmen 179
Close() 229
CMAP-Chunk 235
Compiler 147, 219
Complement-Zeichenmodus 150
Computergrafik 97
Computermodel 11
Consol-Fenster 135, 148
CPU 70
CRNG-Chunk 235

D

Datenstruktur 100, 184
Default-Wert 149
Deffinitionsbereich 41
Definitionsfehler 210
Deluxe Paint 151
Demodiskette 250
Determinante 183
Diagramm 20
Differential-Geometrie 126
Differentialgleichnung 268
Dimension, fraktale 268
Directory 147
Diskette 133, 235
Divergenz 268
–, Bereich 36, 47
Division 40, 40
Dokumentation 201
DOS 226
–, Funktionen 231
Double-Buffered-Display 187
Drachen-Kurve 108, 111
Drehsymmetrie 60
Drehung 175
Dreieck 100
Durchläufe 47

E

Ebene, komplexe 60
Echtzeitberechnung 71
Echtzeitfilm 133
Eckpunkte 180
Eingabekanal 207
Eingangsparameter 36
Einheitswurzel 80
Einschwingvorgang 21
Einzugsbereich 86
Einzugsgebiet 268
Electronics Arts 232
Endmodell 102
Entwicklungsgrenze 12, 12
Erweiterungsfunktion 151
Euler-Verfahren 34
Eulersche Gleichung 40
Exclusive Write Lock 227
Execute-File 253
Experimentielle Mathematik 12
Exponent, gerader 66
–, ungerader 66
Exponentialfunktion 40
Expotentialgesetz 11
Extra-Half-Brite-Modus 151
Extra-Menü 149

F

Farbregister 21, 24
FAST-Memory 133
Fatou, Pierre 52, 268
Feigenbaum, Mitchell 18, 29, 268
Feigenbaum-Diagramm 8, 11, 29
Feigenbaum-Formel 34
feigenbaum1.c 18, 251
feigenbaum2.c 22, 251
feigenbaum3.c 24
feigenbaum4.c 27, 124, 251
feigenbaum5.c 32, 252
Feldverwaltung 102
Fenster 207
File-Struktur 228
FileInfoBlock-Struktur 227
film.c 256

Fixpunkt 268
Flächen-Tabelle 185
Flächendefinition 76, 188
Flächenfarbe 184
Flächenmodell 100, 180
Flag 82
Floppy 226
Fluchtpunkt 178
Folge 269
–, rekursive 36, 269
–, zyklische 45, 269
Folgenglieder 16, 94
Folgenverhalten 43
Folgenwerte 18
FORM-Chunk 233, 236
Format-Menü 149
Formenvielfalt 13
Fortschrittsglauben 9
fractal.c 102, 152, 254
Fraktal 127, 269
Fraktal-Generator 70, 76, 147
fraktal.menu 254
Fraktale Grafik 7
frame.c 150, 219, 254
Füllfarbe 221
Füllfunktion 221
Fünfeck 180
Funktion 78, 108
Funktion-Menü 149

G

Gadgets 213
Gedankenexperiment 38
Geisteswissenschaft 12
Geometrie 126
Gerade 20
Geschwindigkeitsdiagramme 48
Geschwindigkeitsoptimierung 50
GetMsg() 207
Gittermodell 256
Gitternetz 102
graf2.c 124
Grafik, dreidimensionale 99
Grafik-Library 218
Grafikfunktionen 218, 231
Grafiksystem 97
Grafische Auswertung 21
grafl.c 116
Grenze 269
Grenzfigur 36
Grenzverlauf 68, 86
Grenzwert 17, 29, 31, 41, 269
Grundfigur 96
Grundraster 100

H

HAM-Modus 202
Häufigkeiten 21
Hausdorf, Felix 128
Hausdorf-Besicovich-Dimension 127
Henon-Attraktor 36
Herzkurve 51
Hidden-Lines 186, 269
Hilbert-Kurve 108
hilbert.c 109
Hilfswert 91
Höhenlinie 71, 269
Hold-and-Modify-Modus 202
Hyperbel 29
Hypozykloide 51

I

IDCMP-Flag 207, 212
IFF-File 233
IFF-Format 135, 232
IFF-Modul 135
iff.c 253
ILB;-File 234
ILBM-Format 148
Image-Struktur 215
Imaginärachse 60
Imaginärteil 269
Include-File 147
Initialisierung 111, 134
IntuiText-Struktur 215
Intuition 201
–, Library 210
Inversion 176
Item 215
Iteration 31

J

Julia, Gaston 52
Julia-Menge 8, 18

K

Kanten-Tabelle 185
Kardioide 51
Kaskadenverzweigung 20, 270
–, doppelte 29
Kausalitätsprinzip 270
Koch-Kurve 108
Kochsche Schneeflocke 129
konvergentes Verhalten 46
Konvergenz 41
Konvergenzbereich 29, 36, 42
Koordinaten, polare 270
–, relative 270
Körper-Tabelle 185
Körpersystem 177
kreis.c 256

L

Laderoutinen 135
land.c 104, 255
Landschaften, künstliche 8
Landschaftsmodell 102
lands.menu 103
Lattice C 147, 249
Laufrichtung 108
Laufwerk 226
Lichtquelle 181
Linker 148
Literaturverzeichnis 259
Lock-Struktur 227
LOGO 126
lokale Variable 109

M

Magnetismus 96
Makro 216, 222
Malprogramm 232
Mandelbrot, B. 7, 13
Mandelbrot-Ebene 18
Mandelbrot-Folge 45
Mandelbrot-Menge 13, 46
Mandelbrot-Verfahren 68
Mathe-Library 148
Mathematik 7, 36, 38
–, experimentelle 271
Mathematik-Routinen 148
Matrix 174
–, inverse 176
Matrixmultiplikation 271
Matrixoperation 175
Matrizenmultiplikation 199
Matrizenrechnung 175
Mausposition 211
Maustaste 150
Mengen höherer Ordnung 63
Menü-Funktionen 216, 231
menu.c 216, 254
MenuItem-Struktur 214
Modellgleichung 96
Modulo-Funktion 21, 117
Moivre-Formel 41
Monitor 10
MoveWindow() 211
Multiplikation 39
Multitasking 201

N

Näherung, grobe 30
Näherungs-Verfahren 78
Näherungsverfahren 34
Naturerscheinungen 9
Naturkonstante 30, 268
Naturwissenschaft 7
Newton-Verfahren 13, 42, 91
newton.c 251
NewWindow-Struktur 150
Normalvektor 183
Nullstellen 78
–, Suche 78

O

Objekt-Datei 2409
Objekt-Modul 253
OpenScreen() 203
openScreen.c 206
Optimierung 47
Ordnung, innere 90
Ordnungsprinzipien 9

P

Packungsdichte 232
Parabel 30
Parameterwahl 16, 68
Periodenverdopplung 271
periodisches Verhalten 63
Perspektive 71, 173, 177
Phase 39
Pixel 186
Polarkoordinaten 39, 173
Polygone 220
Population 10, 16
present.c 253
pro.c 71
Progammdiskette 142
Programmierfehler 179
Programmierung 147
Projekt-Menü 148
Projektion 177
Projektor 133
Prozessor-Struktur 228
Pseudo 3-D-Grafik 186
Public-Domain-Programm 133
punkt.c 254
Punktdefinition 76, 188
Punktsymmetrie 60

Q

Quadrant 30
Quadratische Ergänzung 37

R

RAM 147
–, Disk 226
Randbedingung 34
Randstruktur 43
RasInfo-Struktur 134
Raster 22
Rasterpunkt 101, 186
Rasterung 186
RastPort-Struktur 134
Raumlage 175
räumliches Gebilde 24
Realitätsnähe 182
Realteil 271
Rechenaufwand 82
Rechengenauigkeit 148
Rechenzeit 69, 90
rechteck.c 254
Reflexionsverhalten 182
Register 102
Rekursion 109
rekursive Folge 13, 36, 42
–, Formel 16
REM-Zeile 188
ReplyMesg() 207
Requester 213
Richtungsvektor 182
Rollkurve 50
Rückkopplung 41, 57
Rückkopplungsformel 60

S

save() 82
Schattierung 181
Schildkröten-Grafik 108
Schmetterlingseffekt 12
Schnitt 62
Schnittstelle 201
Screen-Funktionen 230
screen.c 134, 252
Screens 201
Selbstähnlichkeit 11, 12, 21, 27
Shared Read Lock 227
showInfo.c 255
SizeWindow() 211
Skalarprodukt 182
Skalierung 177, 271
Software-Enwickler 232
Speicher 228
Speicherbereich 228
Speicherplatz 186
–, mangelnder 210
Spiegelung 60
Startadresse 228
Startparameter 114
Startpopulation 15
Startwert 31, 43
Streckenzug 126
Subtraktion 39
sullivan.c 251
Symmetrie 60
System, dynamisches 7, 36
–, mechanisches 11
–, komplexes 8
Szene 272

T

Tangens 39
Tangente 79
Tasten-Funktionen 188
Technik 38
Teilungspunkt 272
Text-File 253
Text-Gadget 148
TextAttr-Struktur 216
Titeltext 209
Topologische Dimension 127
Transformations-Algorithmus 173
Translation 175
Tripos 227
Turtle-Grafik 107

U

Untersuchungskriterien 43

V

Variable 38
Vektor 38, 183, 272
Vektorprodukt 183
Vektorrechnung 182
verdeckte Linien 179
Verhulst-Attraktor 30, 34
Verhulst-Formel 30
Verhulst-Gleichung 32
Verhulst-System 30
Verkrumpelung 272
Verzerrungen 72
Verzweigung 29, 49
Viereck 100
View-Port-Struktur 134
View-Struktur 134
Vorgängerwert 35

W

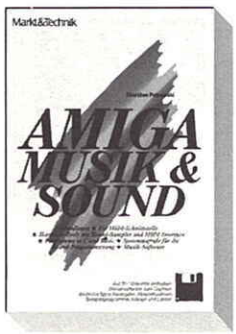
Wachstumsrate 15, 17
WbenchToBack() 205
WBenchToFront() 205
Weltkoordinatensystem 177
Wertebereich 43, 148
Windo-Funktionen 230
Windows 201
WINDOWSIZING-Flag 210
WindowToBack() 211
WindowToFront() 211
Winkel 39, 173
Workbench-Screen 148
Würfel 180
wurz.c 251
Wurzel, komplexe 272

Z

Zahl, komplexe 37
–, reele 37
–, natürliche 41
Zahlenart 38, 38
Zahlenebene 43, 257
Zahlenebene, komplexe 13, 47
Zahlenfolge 272
Zahlengerade 38
Zahlenstrahl 42
Zeichen-Modus 219
Zufallserscheinung 36
Zykloide 50

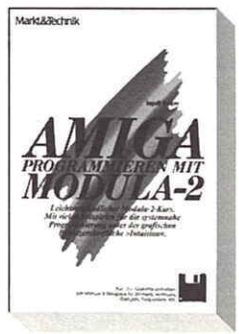
3-D-Grafik 76
–, System 184
68000er 228

Bücher zum Amiga



T. Petrowski
Amiga: Musik & Sound
2. Quartal 1988,
ca. 300 Seiten, inkl. Diskette
»Musik & Sound« bietet zu dem faszinierenden Thema der Musik auf Computern sowohl Anfängern als auch Profis nützliche Tips, Programme und Hardware-Tools. Beispielprogramme und Utilities in Basic und C geben dem Programmierer nützliche Hilfen. Auch die Beschreibungen der Routinen zur Sprach- und Musikprogrammierung unter Basic und C fehlen nicht, und ein Extrakapitel geht auf MIDI und deren Programmierung ein.
● Einführung in das faszinierende Thema Musik auf Amiga-Computern.
Bestell-Nr. 90551
ISBN 3-89090-551-4
ca. DM 69,–
(sFr 63,50/öS 538,20)

P. Wollschlaeger
Amiga-Assembler-Buch
1987, 329 Seiten, inkl. Disk.
Nach einem Minimum an Theorie geht dieses Buch sofort in die Praxis. Aus dem Inhalt: Grundlagen des 68000er, Systemprogrammierung, Programmierung von Intuition, schnelle Grafik in Farbe, alle Systemroutinen mit Parametern.
Bestell-Nr. 90525
ISBN 3-89090-525-0
DM 59,–
(sFr 54,30/öS 460,20)



I. Krüger
Amiga: Programmieren mit Modula 2
1988, 350 Seiten, inkl. Disk.
Leichtverständlicher Modula-2-Kurs! Mit vielen Beispielen für die systemnahe Programmierung unter der grafischen Benutzeroberfläche »Intuition«. Aus dem Inhalt: Programm-Module, Variablendeklaration, Strukturanweisungen, Prozeduren, lokale und externe Module, Verwendung von Zeigern, systemnahe Programmierung, Coroutinen (Verarbeitung von parallelen Prozessen), Programmierung unter Intuition (Screens, Windows, Gadgets, Requester).
Bestell-Nr. 90554
ISBN 3-89090-554-4
DM 69,–
(sFr 63,50/öS 538,20)

Markt&Technik
Zeitschriften · Bücher
Software · Schulung

Markt&Technik-Produkte erhalten Sie bei Ihrem Buchhändler, in Computer-Fachgeschäften oder in den Fachabteilungen der Warenhäuser.

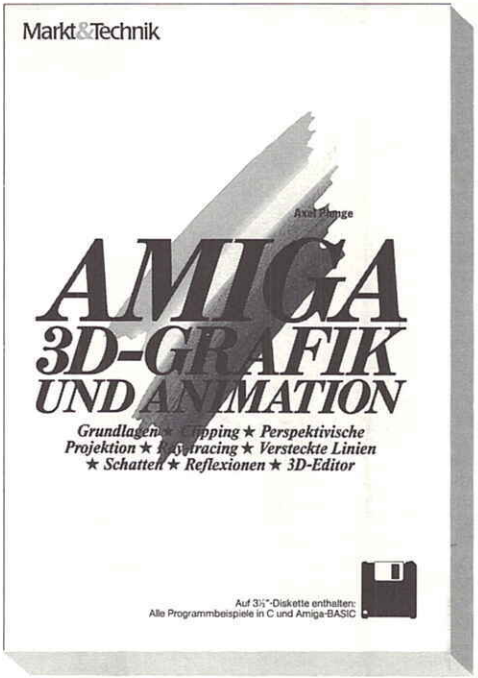
804331

Markt&Technik Verlag AG, Buchverlag, Hans-Pinsel-Straße 2, 8013 Haar bei München, Telefon (089) 4613-0

Bücher zum Amiga

M. Breuer
DELUXE
Grafik mit dem Amiga
1987, 370 Seiten
Eine behutsame Einführung in die grundlegenden Konzepte des jeweiligen Programms. Führt anhand kleiner überschaubarer Beispiele die wichtigsten Programmbefehle vor. Ein Nachschlageteil zu jedem Programm listet alle Befehle und ihre Bedeutung auf. Den Abschluß der Beschreibung jedes Programms bildet eine Sammlung von Tips und Tricks.
- Das deutsche Handbuch für den kreativen Grafikkünstler mit der DELUXE-Serie.

Best.-Nr. 90412
ISBN 3-89090-412-2
DM 49,–



A. Plenge
Amiga 3D-Grafik und Animation
1988, ca. 350 Seiten, inkl. Disk.
Angefangen bei den einfachsten Problemstellungen lernen Sie, professionelle 3D-Grafiken auf Ihrem Commodore Amiga zu planen, zu programmieren und darzustellen. Sämtliche theoretischen Grundlagen werden Ihnen anschaulich und leichtverständlich vermittelt. Auch scheinbar komplizierte Grafiken wie Schatterbildung, Reflexion, durchsichtige Gegenstände, Vielfachspiegelungen oder »raytracing« werden verständlich dargestellt.
- Eine ausführliche und leichtverständliche Anleitung für die Erstellung von dreidimensionalen Grafiken.

Best.-Nr. 90526
ISBN 3-89090-526-9
DM 69,–

M. Kohlen
Grafik auf dem Amiga
1987, 337 Seiten
Dieses Buch enthält eine ausführliche Beschreibung der Grafik-Hard- und -Software, deren Funktionsweise und führt in die Grundzüge der Grafikprogrammierung ein. In den folgenden Kapiteln werden diese Kenntnisse dann in praktischen Beispielen umgesetzt. Außerdem bietet das Buch einen Überblick über die vorhandenen Soft- und Hardware-Erweiterungen für den Amiga.
- Eine Pflichtlektüre für jeden, der sich für die phantastische Grafik des Amiga interessiert.

Best.-Nr. 90236
ISBN 3-89090-236-7
DM 49,–

Markt&Technik
Zeitschriften · Bücher
Software · Schulung

712300

Markt&Technik Verlag AG, Buchverlag, Hans-Pinsel-Straße 2, 8013 Haar bei München, Telefon (089) 4613-0

Bitte schneiden Sie diesen Coupon aus, und schicken Sie ihn in einem Kuvert an:
Markt&Technik Verlag AG, Buchverlag, Hans-Pinsel-Straße 2, 8013 Haar

Computerliteratur und Software vom Spezialisten

Vom Einsteigerbuch für den Heim- oder Personalcomputer-Neuling über professionelle Programmierhandbücher bis hin zum Elektronikbuch bieten wir Ihnen interessante und topaktuelle Titel für

• Apple-Computer • Atari-Computer • Commodore 64/128/16/116/Plus 4 • Schneider-Computer • IBM-PC, XT und Kompatible
sowie zu den Fachbereichen Programmiersprachen • Betriebssysteme (CP/M, MS-DOS, Unix, Z80) • Textverarbeitung • Datenbanksysteme • Tabellenkalkulation • Integrierte Software • Mikroprozessoren • Schulungen.
Außerdem finden Sie professionelle Spitzen-Programme in unserem preiswerten Software-Angebot für Amiga, Atari ST, Commodore 128, 128 D, 64, 16, für Schneider-Computer und für IBM-PCs und Kompatible!
Fordern Sie mit dem nebenstehenden Coupon unser neuestes Gesamtverzeichnis und unsere Programmservice-Übersichten an, mithilfreichen Utilities, professionellen Anwendungen oder packenden Computerspielen!

Markt&Technik
Zeitschriften · Bücher
Software · Schulung

Markt&Technik Verlag AG, Buchverlag, Hans-Pinsel-Straße 2,
8013 Haar bei München, Telefon (089) 4613-0

709005

Adresse:

Name

Straße

Ort

Bitte schicken Sie mir:
☐ Ihr neuestes Gesamtverzeichnis
☐ Eine Übersicht Ihres Programmservice-Angebotes aus der Zeitschrift _____

☐ Außerdem interessiere ich mich für folgende/n Computer: _____

[PS.: Wir speichern Ihre Daten und verpflichten uns zur Einhaltung des Bundesdatenschutzgesetzes]

Markt&Technik Verlag AG
– Unternehmensbereich Buchverlag –
Hans-Pinsel-Straße 2
D-8013 Haar bei München

Amiga-Software

CLImate 1.2

Jetzt stehen Ihnen die Funktionen Ihres Amiga-Command-Line-Interface per Mausklick zur Verfügung!

Mit diesem Programm können Sie die Befehle des Command-Line-Interface (CLI) benutzerfreundlich und schnell per Mausklick verwenden!

Ihre Super-Vorteile mit CLImate 1.2:

- sehr große Übersichtlichkeit der Bildschirmdarstellung (Sie haben alle Funktionen auf einen Blick)
- leichte Bedienung aller Befehle mit der Maus
- drei externe Laufwerke (3½" oder 5¼"), zwei Festplatten, RAM-Disk unterstützen Sie
- schnelle Directory-Anzeige
- Sie können Disketten leicht nach Texten, Bildern u.ä. durchsuchen
- Dateien lassen sich mit Pause/Continue-Möglichkeit betrachten
- Ausdrucken von Dateien auf Drucker
- Informationen über die Disketten (Programmlänge und ähnliches)
- Betrachten von Bildern im IFF-Format (inklusive HAM)

Bestell-Nr. 51653

DM 79,-*
(sFr 72,-*/öS 990,-*)
*Unverbindliche Preisempfehlung

- Sie können Dateien aus beliebigen Verzeichnissen in andere Verzeichnisse kopieren
- Bildschirmausgabe von Dateien in ASCII und in hexadezimaler Form
- Unterstützung von Jokerzeichen bei Disketten- und Dateioperationen

CLImate 1.2 - das unentbehrliche Programm für den Amiga-500-, Amiga-1000- und Amiga-2000-Besitzer.

Am besten gleich bestellen!

Hardware-Anforderungen: Amiga 500, 1000 oder 2000 mit mindestens 512 Kbyte Hauptspeicher. Empfohlene Hardware: Farbmonitor. Software-Anforderungen: Kickstart 1.2 (oder ROM bei Amiga 500 und 2000), Workbench 1.2. Eine 3½"-Diskette für den Amiga 500, 1000 und 2000.

Markt&Technik
Zeitschriften · Bücher
Software · Schulung

Markt&Technik-Produkte erhalten Sie bei Ihrem Buchhändler, in Computer-Fachgeschäften oder in den Fachabteilungen der Warenhäuser

801271

Markt&Technik Verlag AG, Buchverlag, Hans-Pinsel-Straße 2, 8013 Haar bei München, Telefon (089) 4613-0

Fraktale Grafik auf dem Amiga

Der Autor:
Heiko Knappe, Jahrgang 1965, studiert im siebten Semester Elektrotechnik an der Technischen Universität München mit Schwerpunkt Informationsverarbeitung. Er beschäftigt sich seit vielen Jahren mit Computern, insbesondere mit dem Amiga. Neben seinem Studium veröffentlichte er bereits mehrfach als freier Autor Artikel über die Programmiersprache C, Programmieren mit dem Amiga und über die Amiga-Hardware.

Meist wird die Mathematik als eine starre Wissenschaft betrachtet, in der es nichts mehr zu entdecken gibt. Daß dies nicht stimmt, zeigt die experimentelle Mathematik. Hier findet ein Umdenkungsprozeß statt, seit B. Mandelbrot seine »Theorie der Ordnung im Chaos« erstmalig vorgestellt hat und es ihm gelang, diese mit Hilfe von Fraktalen mathematisch zu beschreiben. Dieses Umdenken betrifft auch die anderen Naturwissenschaften und ist eigentlich für die gesamte Weltbetrachtung von Bedeutung. Wurde bisher davon ausgegangen, daß bei genügender Kenntnis die Welt deterministisch sei, versteht man heute langsam, daß nicht die Ordnung, sondern das Chaos das in der Natur bestimmende Element ist.
Gerade Fraktale eignen sich wie kaum ein anderes Gebiet der Mathematik für die Bearbeitung mit dem Computer. Hier findet auch der Laie schnell Zugang über die »innere Schönheit« dieser bizarren grafischen Gebilde. Bekannte Entstehungsprozesse können leicht nachvollzogen werden, um schließlich immer neue Variationen des »Chaos« zu verstehen und zu erforschen.
Mit dem Amiga haben wir ein ideales »Labor« für diese Experimente zur Verfügung. Seine schnelle CPU, sein ausgereiftes Betriebssystem und seine hochauflösende Bildschirmgrafik machen ihn zu einem leistungsfähigen Grafiksystem der unteren Preisklasse.
Dieses Buch soll den Weg zum eigenen Experiment vereinfachen – es beinhaltet deshalb eine Vielzahl an Beispielprogrammen, die ohne großen Aufwand leicht für eigene Problemstellungen modifiziert werden können.
Grundlegende Kenntnisse der Programmiersprache C und ihrer Programmierung auf dem Amiga werden vorausgesetzt.

Aus dem Inhalt:

- komplexe und rekursive Folgen
- Feigenbaum-Diagramme
- mathematische Grundlagen
- komplexe dynamische Systeme
- Julia-Mengen
- Apfelmännchen
- künstliche Landschaften
- Grundlagen dreidimensionaler Grafik
- C-Programmierung auf dem Amiga
- Betriebssystemroutinen
- IFF-Routinen
- Einführung in Intuition

und vieles mehr

Die Begleitdiskette:

- enthält alle im Buch beschriebenen Programme im Quellcode

Hardware-Anforderungen:

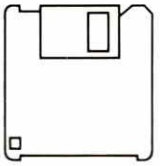
- Amiga 500, 1000 oder 2000 (eventuell mit Speichererweiterung und Drucker)

Software-Anforderungen:

- C-Compiler – Aztec C oder Lattice C (eventuell Malprogramm z.B. Deluxe Paint)

ISBN 3-89090-600-1

Markt&Technik



DM 79,-
sFr 72,20
öS 616,20